2003-09

Documentation Driven Agile Development for Systems of Embedded Systems

Luqi; Zhang, Lynn

Documentation Driven Agile Development for Systems of Embedded Systems, with L. Zhang, Monterey Workshop Series: Workshop on Software Engineering for Embedded Systems: From Requirement to Implementation, Chicago, IL, pp.13-25.
https://hdl.handle.net/10945/42337

Documentation Driven Agile Development for Systems of Embedded Systems

Luqi, Lynn Zhang

Software Engineering Automation Center
US Naval Postgraduate School
{luqi; lzhang} @ nps.navy.mil

Abstract: This paper presents the framework of documentation-driven agile development (DDAD) methodology for high confidence systems of embedded systems. DDAD mainly includes two parts: a documentation management system (DMS) and a process measurement system (PMS). DMS will create, organize, monitor, analyze and transform all documentation associated with the software development process. The information will be stored in an abstract and active form that will support a variety of formal and informal documents for different stakeholders and can interact with software tools. PMS will monitor the frequent changes in system requirements and assess the effort and success possibility of the project with a measurement model based on a set of quantitative metrics that can be automatically collected in requirements phase and stored and organized in DMS. PMS will also measure the properties of the software system that must be realized with high confidence (safety in this paper) based on quantitative metrics. DDAD will provide a mechanism to monitor and quickly respond to changes in requirements and provide a friendly communication and collaboration environment to enable different stakeholders to be easily involved in development processes and therefore significantly improve the agility of software development of SoES. DDAD will also support automated software generation based on a computational model and some relevant techniques. Several potential application domains are proposed in the paper.

Keywords: Software Development; Documentation; Agility; Knowledge Representation; Systems of Embedded Systems.

1. Introduction

Design of real-time embedded systems involves a multi-disciplinary team of systems, software and hardware engineers. They have different concerns, use different tools, and work somewhat independently of one another. For a high confidence system of embedded systems, development is much more complex than development of monolithic embedded systems. Non-essential software complexity of a system of systems can have a greater negative impact on system behavior than for a single system. In general, systems of embedded systems are usually deployed for long periods of time, are used globally, and have mission critical requirements. They demand real-time performance and high confidence. Attributes like system effectiveness, availability, reliability, safety, security, and clarity of design are all essential. Most importantly, the SoES must rapidly accommodate frequent changes in requirements, mission, environment, and technology. Consequently they are often structured as a coalition of separate components to form systems of embedded systems with dynamic configurations. In addition, SoES are usually composed of component systems that were developed by different organizations with different tools and run on different platforms. A wide variety of stakeholders (sponsors, developers, users, maintainers, etc.) are involved in the overall lifecycle of the software [1, 20].

A large amount of research has been conducted on real-time systems. Progress has been made, but mostly on "point solutions" that address sub-areas of complex system development. Integrated systematic methods that collectively provide an end-to-end solution, are easy to use, and are amenable to computer aid are needed to meet these challenges.

Software development agility is drawing more and more attention in the software engineering community. Agile software development is presented as the solution to deal with the frequent changes of requirements [11]. This approach focuses on individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; responding to change over following a plan [37]. Thus, compared to other methods heavily depending on the traditional documentation, many current agile software development methods try to provide better communications

with the user, reduce the comprehensive documentation and be capable to adapt to requirements changes. Some typical agile development methods are extreme programming (XP); dynamic software development method (DSDM); adaptive software development; feature-driven development; lean development; rapid application development etc.

Extreme Programming (XP) was created in response to problem domains whose requirements change [8, 38]. The XP practices are also intended to mitigate the risk and increase the likelihood of success. XP requires an extended development team. The XP team includes not only the developers, but also the managers and customers, all working together elbow to elbow. Asking questions, negotiating scope and schedules, and creating functional tests require more than just the developers be involved in producing the software. However, XP is only suitable for small groups of programmers, between 2 and 12. XP was not designed for a project with a huge staff or a large number of different stakeholder roles.

DSDM uses an iterative process based on prototyping and involves the users throughout the project life cycle [9]. DSDM achieves delivery with tight timescales through shortening communication lines between users and developers, between analysts and designers, between and across team members, and between differing levels of management. The mechanisms by which these communication lines are shortened differ from one application to another. DSDM defines a strategy for defining what the necessary documentation set will be for a given project. Much of the documentation that is traditionally produced is for the transfer of ideas from one developer to another or from developers to users. DSDM provides guidance on how to decide what sort of documentation is necessary and why. There are key criteria that a project should satisfy for DSDM to be applied easily. The project should be able to identify all the classes of users who will use the end result so that knowledgeable representatives can participate throughout the lifecycle of the project and provide coverage of the views of all the user classes.

These agile methods' attitude to documentation is to reduce the amount of traditional informal documents as much as possible by increasing direct communications between users and developers. The problem with these approaches is that the users are required to be knowledgeable and well versed in the software domain skills to be able to participate in the development process. Following some of the agile principles runs a high risk when the motivated individuals don't have the requisite domain skills [39]. Moreover, software development automation is reduced when direct communications between users and developers are over emphasized. It's well known that the automation of development can significantly improve productivity and minimize errors in software products. A good tradeoff between software development automation and agility is needed to develop systems that require high confidence on a large scale with frequent changes.

Making suitable use of documentation in the development process can reduce the requirements for participants to have specific knowledge. Moreover, by generalizing and abstracting the essence of documentation and exploiting the capability for computer-aided documentation, documentation can be used to significantly improve the agility of SoES software development while sacrificing automation to a minimum extent.

According to traditional concept and current common practice, software documentation consists only of informal text and diagrams intended for human consumption. This kind of static information in documentation cannot provide effective support for the development process, especially for systems of embedded systems. In our opinion, this traditional concept should be extended so that all the information needed to carry out the development process is considered documentation. The requirements for both high confidence and frequent changes in systems of embedded systems can only be realized by development processes that provide effective computer aid. Effective documentation should support humans to the extent the relevant development processes are carried out by humans, and should support software tools to the extent development processes are carried out by tools. In the common case where an aspect of the development process is carried out by a collaboration of both humans and software tools, the documentation should provide two views, one for the humans and one for the tools. For such aspects, consistency and accurate correspondence between the two views are of most importance, and computer aid is needed to effectively realize these properties.

In this approach, models and simulations are included as documentation. Some typical models include computational models and design models. They serve as the basis to support development activities such as requirements analysis, architecture design, validation and verification. Simulation and prototyping are examples of computer aided processes used to check the correctness of the requirements for the system under development. With this extension, documentation can provide more effective support for whole development process. This paper proposes a documentation driven methodology with respect to the features of systems of embedded systems. This methodology will significantly improve the agility of software development to accommodate frequent changes in requirements of SoES and support partial automation of software development as well.

2. Overview of Documentation Driven Agile Development

Agile development emphasizes the relationship and cooperation of different stakeholders. It requires that the development group, comprised of system designers, hardware developers, software developers and customer representatives, should be well-informed, competent and authorized to consider possible adjustment needs emerging during the development process life-cycle [26]. Our idea to improve agility on a large scale by taking advantage of a good documentation system is depicted in Figure 1. It's named the Documentation Driven Agile Development (DDAD) methodology. Three typical development processes are shown to illustrate the methodology.

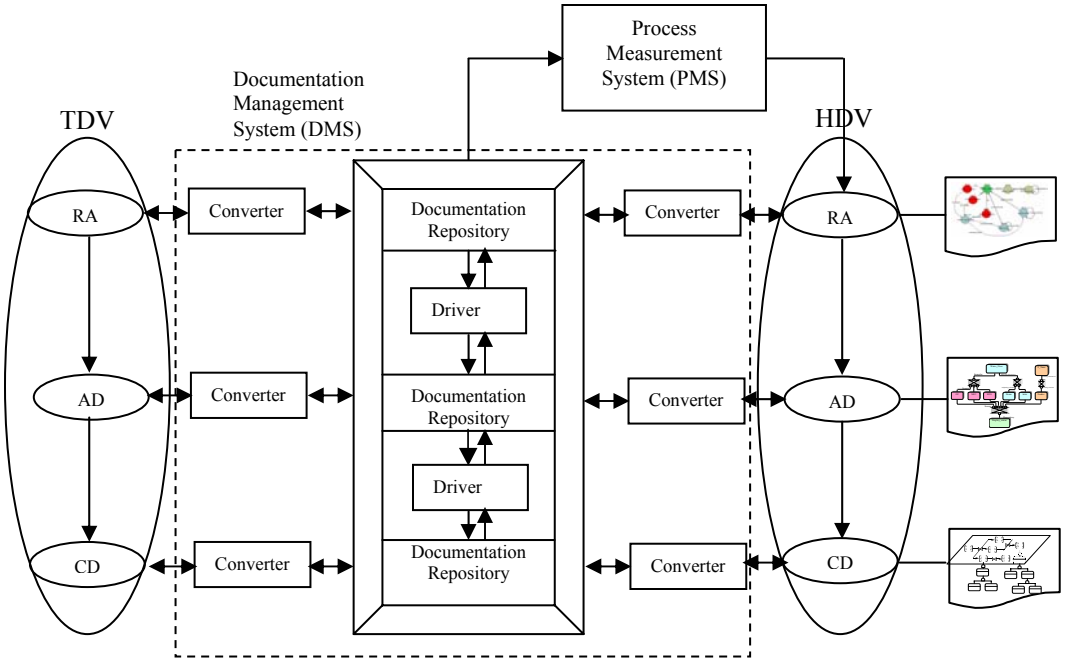

Figure 1. Documentation Driven Agile Software Development

The main idea behind DDAD is to build and use a Document Management System (DMS) and a Process Measurement System (PMS). The key to DDSD is that information from any activity involved throughout the software development process as well as the entire software life cycle will be recorded, managed and transformed by the DMS. The information will be stored in a form that will support a variety of formal and informal documents for different stakeholders and can be manipulated by a set of software tools. Eventually, the DMS will monitor and drive the overall development process and be applied throughout the entire software life cycle. DMS makes the development processes transparent and traceable, enables documentation to be updated quickly and facilitates communications and collaboration between stakeholders to promptly respond to changes in requirements. Process Measurement System (PMS) is used to track and analyze changes in requirements to verify the feasibility of the requirements, assess effort and

risk of development, provide clues to modify the requirements, and measure the required high confidence properties. PMS is based on a set of quantitative metrics, most of which can be automatically collected in requirements phase. These metrics are stored and organized in the documentation management system. PMS and DMS working together will help the development of SoES rapidly accommodate frequent changes in requirements.

3. Documentation Management System (DMS)

DMS will create, organize, monitor, analyze and manipulate all documentation associated with the software development process. It will record all information from the development process such as requirement specifications, abstracted models, stakeholder input, design rationale, project management information and the source code. It will also extract important information from all development activities such as requirements analysis, prototyping, architectural design, software composition, system verification and validation, and system deployment. A documentation repository will be used to store the information in a structured, well-organized format. Information from the repository will support knowledge transfer between processes and generate the various presentations of this information for the different stakeholders and tools. The information stored in the repository drives both the Tool Documentation View (TDV) and Human Documentation View (HDV). By doing this, the development processes can be automated and the communications between stakeholders can be easier.

Tool Documentation View (TDV) representations are based on formal representations of the knowledge stored in the documentation repository and transformed into a format appropriate for use by the computer environment (software tools). They are usually in the form of mathematical formulas like temporal logic or process algebra, formal languages like PSDL or ADL, and programming languages. Typical TDVs include system models, requirements/design specifications, ontologies, source codes, test cases etc. They can also include application data such as geographic databases, results of measurements, medical records, financial databases, tables of properties of physical materials, and any other reference information relevant to system design.

Human Documentation View (HDV) representations are typically graphical in nature and in a form easily understood by humans. They are used by the stakeholders to communicate and interact with each other (sponsors, end users, developers (system, hardware and software engineers), technical supporters, etc.). Additional forms include text annotations written in natural language, decision tables and spread sheets. They can easily be expanded to include modern communication techniques such as video and audio clips. The latter can be useful for recording raw data about application process and content, to capture implicit requirements information that system stakeholders can demonstrate but cannot describe. The information in the HDV can include computed attributes that are not explicit in the information entered into the DMS. We envision this type of information to be useful for engineering and project management decision support. Examples include results of design rule checks, values of performance and reliability metrics, projections of project completion date and cost, and project risk metrics.

DMS contains a set of tools (e.g. converters and drivers) that will automatically convert the stored information from one representation to another to support different stakeholders and integrate the development processes by driving the knowledge transfer between them.

3.1 Documentation Repository

Keeping documentation up to date is difficult because of the various representations of information used in various stages of the development. The various representations of the same documentation information increase the complexity of maintaining information consistency and also hinder unaided communications between human and machine. Although multiple views of the information can solve this problem, how to maintain consistency among information presented to both the human and computer tools is still a challenge. This paper presents a documentation repository in which a common internal representation, template-based knowledge representation, is used to represent all information contained in the documentation.

Template-based knowledge representation is the kernel part of the documentation repository. It includes the following artifacts:

- Document Elements that are described by a semantic document model. It is an object model for the information contained in the documentation whose instances form an attributed object graph.
- A set of syntactic templates. The specifying elements together with syntactic templates can translate representations from one form to another or transform the information from one view to another.
- Attribute computation rules. This artifact represents the methods for computing derived document attributes.

Document Element

A document element is a basic building block consistent with the semantics of the information contained in the documentation. We use a semantic model named Attributed Object Graph Model (AOGM) to describe the semantics of each document element [16]. This is an object model of knowledge in the documentation repository. It has a nested structure with potentially shared nodes, i.e., directed acyclic graph structure. This representation is a generalization of abstract syntax trees that was developed in our previous research to represent constructs that appear in more than one context. This is a common pattern in software artifacts – for example, an operation can be defined once and called from many different contexts. In this model, each node represents a semantically meaningful structure, such as an individual requirement, a subsystem, an operation, or an operator within a logical expression. The nodes are the finest grain structures visible to the attribute computation rules. Furthermore, each node is an instance of an abstract data type. The computed attributes of each node correspond to the operations of the data type. Thus, invoking appropriate methods of the data type can derive the value of the corresponding attributes.

Syntactic Template

To improve the communication between the human and machine during the development process, computed multiple views of the same information for different people and different computer tools involved in the development provide a way to avoid inconsistencies between different representations of the same information due to incomplete manual updates. We are developing corresponding templates to support multiple views of the information. These views include the Human Document View (HDV) and the Tool Document View (TDV). In this case, the templates serve to transform the information from one view to another.

Syntactic templates are object operations with parameters. They provide a context for the resident document elements that will appear in different kinds of specifications. The combination of a document element and its syntactic context forms the multiple view presentation for the same information. Combining document elements with corresponding templates can also transform the information between representations written in different description languages.

We use tokens in an initial prototype representation of templates. Special tokens such as blank-filling tokens and action-interpreting tokens support computation of concrete document views. The blank-filling tokens indicate the blanks to be filled out, the actions to be interpreted and the information to be correlated etc. Action-interpreting

Template Items	Formalized Identification	Operational Semantic
Key-word	≪! key !≫	Key word to be matched
Token-Blank	≪@type@≫	Type to be replaced with the value of a document element
Token-In / Token-Out	≦ … … ≧	Enclosed by Token-in and Token-out will be contributed as properties of preceding Token
Routine Action	≪&action&≫ ≪&NL&≫ ≪&HL&≫ … …	Action to be performed New line is output Hyper Link is followed … …
Appearance of N≧0	*⌈ … ⌋	Items that appear 0 to n times
Appearance of 0 or 1	∘⌈ … ⌋	Items that appear once or none
Selective Appearance	⌈ <condition1> -> <item1> ⇡ <condition2> -> <item2> ⇡ … ⌋	Select one of values from list
Semantic symbol	≪, ≪@, ≪&, *⌈, ∘⌈ , ⇡ ≫, @≫, &≫, ⌋, ≦, ≧	Enumerated characters have special meanings for software tools
Real Appearance	Typed characters	Any character appearing in the template only represents itself
Template Comment	//	Omitted

Table 1. List of Semantic Tokens

tokens are used to indicate actions to be conducted by software tools. Some possible tokens are listed in Table 1.

Attribute Computation Rules

We are studying methods for computing derived attributes and developing a set of schemata used to (a) calculate the attributes from the information in the documentation repository, (b) transform the information from one stage to another, (c) analyze the consistency between the information transformed between stages, and information views, and (d) extract subsets of documents needed for particular purposes.

Based on the Attributed Object Graph Model (AOGM), we developed a set of attribute rules to check whether significant aspects of the meaning are preserved during the information transformed from one development phase to another phase. These attribute rules can ensure that there is no information lost in transformation. We used timing properties transformation between requirement phase and design phase as the example to describe corresponding attribute computation rules [16].

3.2 Representation Converter

The representation converter presents the repository documentation to different stakeholders in a traceable, consistent and understandable way. These presentations include graphical depiction, formal description, logic formulation, audio and video media and so on. This tool will present the knowledge embodied by specifying elements and syntactic templates in a form the stakeholders can understand. The converter is based upon the combination of the knowledge-centric templates and the collection of specifying elements. It will "combine" the content of the document elements and the syntactic templates together to create and present desired documents for different stakeholders. Based on a specific template design, the tool generates presentation output for different stakeholders. A template selector is used to determine what kinds of documents will be produced. Also, based on the specific template design, the converter guides information to a collecting specifying element. This is similar to drag and drop with dialogue resources supported in a Windows application.

We have conducted research on a successful example that supports multiple document presentations based upon syntactic knowledge, such as the Computer Aided Prototyping System (CAPS) [17, 18, 40]. CAPS is the computed-aided prototyping system, whose computational model can be described in both PSDL specification and graphical depiction. Different stakeholders can share this information. Although a designer will use both the formal and graphical documents, a customer might use just a graphical document, and software tools use just the formal documents.

3.3 Transition Driver

A transition driver serves as a process transition tool based on the combination of knowledge-centric templates and a collection of document elements. Its function is to analyze the key information held by the templates and the document elements and to promote the transition of repository knowledge from one development process to the next. A transitional driver has the ability to act in both a forward and reverse direction. It can drive the transition of knowledge from one process to a succeeding one (forward) or from one process to a preceding one (reverse). In the first mode, the transitional driver promotes forward engineering of software products. The transition driver analyzes the preceding knowledge (knowledge used as an input), guides user's intervention, and then generates succeeding knowledge (process output). In the second mode, the driver promotes reverse engineering of legacy software systems if necessary. In this case, the driver serves as an extractor. It performs analysis and extracts useful information from what is normally considered the output information from a phase and generates what should have been the input information for that phase. A challenge in this area is how to best manage designer and user interaction to extract specification and design information the way it should have been built, rather than capturing the way it actually was built, including all of the errors and faults. A first step is to support annotations that identify such faults with links to explanations of why they constitute faults.

4. Process Measurement System (PMS)

The function of the process measurement system is to monitor the frequent changes in system requirements, assess the effort and success possibility of the project, and measure the high confidence properties of the system. The PMS obtains necessary information from the documentation repository. The analysis results will be presented to the developers and users as feedback. This quick communication is a key factor to make development of SoES agile: feedback is most useful when it can be delivered while the relevant aspect of the system is still in the process of being created, rather than after it has been completed and other system decisions have been made based on a faulty version of that aspect.

The process measurement system includes two parts: (1) a measurement model for effort and risk of a project; (2) a measurement model for high confidence. We have introduced a set of metrics to measure the effort and the risk in an evolutionary software project [22]. These metrics can be automatically obtained early in the requirements phase. They accommodate changes in requirements, process, technology, and resources of a project. Based on the set of metrics, a measurement model has been proposed [22]. The result is a statistical model that is used to estimate development effort and risk of failure of the project. The high confidence measurement model in this paper is only focused on software safety, because safety is the most critical factor for many DoD software systems and the state of the art in software engineering lacks a formal method and metric for measuring safety. We developed an Instantiated Activity Model (IAM) that supports a formal approach for safety analysis by providing precise metrics [30].

4.1 The Measurement Model for Effort and Risk of a Software Project

Current state of the art techniques for risk assessment rely on checklists and human expertise. This constitutes a weak approach because different people could arrive at different conclusions from the same scenario. The measurement model we developed for effort and risk is a statistical model based on a set of quantitative metrics. The metrics include requirements volatility, organization efficiency, product complexity, and technology maturity. This model will enable different program managers to derive the same projections on the same software project.

Metrics for Requirements Volatility

Requirement changing is the most significant characteristic for a system of embedded systems. Requirements volatility clearly influences the possibility of project success. From the point of view of the metrics, a change in a requirement can be viewed as a death of the old version and a birth of the new one. The requirements volatility can be obtained from birth-rate and death-rate. Birth-rate is defined as the percentage of new requirements incorporated in each cycle of the evolution process. Death-rate is defined as the percentage of requirements that are dropped by the customer in each cycle of the evolution process. The requirements volatility (RV) is defined as:

$$RV = BR + DR,$$

where, $BR = (NR / TR) * 100\ \%$, $DR = (DelR / TR) * 100\ \%$, NR = number of new requirements; DelR = number of requirements deleted; TR = total number of requirements.

Metrics for Organization Efficiency

The efficiency of the organization can be measured by observing the fitness between people and their roles in the software process. The skill match between the person and the job is required to estimate the speed in processing information and the rate of exceptions, which in turn affect efficiency. Efficiency also depends on many factors like team structure, experience, and tools. Simulations have shown that there exists an easier way to estimate team efficiency by observing the ratio between direct working time and idle time. The team efficiency metric (EF) is defined as:

$$EF = Dwork\% / Idle\% + Dwork\%$$

where Dwork% is the percentage of direct working time; Idle% is the percentage of idle time.

Metrics for Product Complexity

Product complexity is in general a function of the relationships among the components of the product. Hence, it is important to measure the complexity as a predictor. Product complexity is also directly related to the effort needed to develop a product.

Some requirements are difficult for the user to provide and are difficult for the analysts to determine. It's notably the case for real-time systems. The best way to discover these hidden requirements is via prototyping. CAPS is a CASE tool specially suited for this task, which uses the Prototype System Description Language (PSDL) [17-19]. Specifications written in PSDL can be analyzed to compute the complexity. Metrics for complexity can be defined by using a hybrid complexity measure that properly accounts for data flow and the properties associated with each operator and data stream in PSDL. A complex metric FC is defined as follows:

$$\mathrm{FC} = \sum_{i=1}^{n} w(o_i)[dsi(o_i) * dso(o_i)]$$

where, $w(o_i) = 1 + \sum_{k=1}^{m} pw_k * c_{ik}$ is the total property weight of operator o_i. pw_k is the property weight of the k[th] property, with $0 \le pw_k \le 1$ and $\sum_{k=1}^{m} pw_k = 1$. c_{ik} is the property occurrence coefficient, with $c_{ik} = 1$ if operator o_i has property p_k and $c_{ik} = 0$ otherwise. m is the numbers of property types in PSDL. $dsi(o_i)$ is one plus the number of data streams flowing into operator o_i; $dso(o_i)$ one plus the number of data streams flowing out of operator o_i; n is the total number of operators.

Metrics for Technology Maturity

The software industry is characterized by frequent technology changes. A system of embedded systems is usually deployed for long periods of time and is used globally. In the process of evolutionary development of a SoES, the related technologies will change significantly during the period the system is deployed. Generally, the newer the technology is, the more quickly the technology changes. The impact of technology maturity on success of a project, especially for a SoES, is important.

Technology mainly consists of two parts. One is the software technologies that are selected to implement the project. The other is the domain technologies involved in the project. The choice of implementing technologies should be subordinated to the project domain technologies and requirements.

A new technology becomes mature in the process of transition from a scientific discovery to routine engineering practice in product development. Technology transition is referred to as diffusion in the literature. Diffusion is the process by which an innovation is communicated through certain channels over time among the members of a social system. Based on information theory, communication theory, and statistical mechanics, we developed a metric, named 'technology temperature *T*', to measure the maturity of a technology [23].

According to information theory, the quantity of information in an ensemble of possible messages is measured by entropy. A message is made up of sets of terms. In this context, the relevant information is the knowledge about a technology. Following reasoning similar to that used in statistical and condensed particle physics and recalling the standard definition from the thermodynamics, the temperature *T* for technology transition can be defined as follows:

$$\frac{1}{T} = \frac{\Delta S_H}{\Delta n}$$

where, Δn is the change in the number of terms of a message alphabet Ξ. ΔS_H is the change in entropy. The entropy is defined as follows: for the message alphabet Ξ with the given probability mass function

$p(x) = \Pr\{X = x\}, x \in \Xi$, X is a discrete random variable, the definition of information entropy is $S_H(X) = -\sum_{x \in \Xi} p(x) \log_2 p(x)$.

The temperature is measured in "degrees" in a physical system, however, in the context of information degrees can be expressed in information units (bits). The value of *T* represents the maturity of a technology. It's a function with respect to time step [23].

Measurement Model

A Weibull distribution can be used to build the measurement model. The Weibull distribution was originally used to model strength of Bofors's steel, fiber strength of Indian cotton, length of syrtoideas, fatigue life of steel, statures of adult males, and breadth of beans. Many authors have advocated the use of this distribution in reliability and quality control [21, 25]. Others used it to model software life cycles [15]. The three parameter Weibull distribution is defined as follows.

A random variable x is said to have a Weibull distribution with parameters α , β and γ ($\alpha > 0$, $\beta > 0$) if the probability distribution function (pdf) and cumulative distribution function (cdf) of x are respectively:

$$\text{pdf: } f(x) = \begin{cases} 0 & x < \gamma \\ (\alpha / \beta^{\alpha})(x-\gamma)^{\alpha-1} \exp(-((x-\gamma)/\beta)^{\alpha}) & x \geq \gamma \end{cases}$$

$$\text{cdf: } \mathrm{F}(x) = \begin{cases} 0 & x < \gamma \\ 1 - \exp(-((x-\gamma)/\beta)^{\alpha}) & x \geq \gamma \end{cases}$$

where,

- x is the random variable under study. In our context, x can be interpreted as development time.
- α is a shape parameter. It affects the skew of the function. When α = 1, the function reduces to the exponential distribution. The combined effect of α and β controls the variability of the pdf.
- β is a scale parameter that stretches or compresses the graph in the x direction.
- γ is a location parameter that determines the mean of the pdf.

We have conducted a large number of empirical experiments to determine the relationship between the parameters in the above model and the quantitative metrics above [22]. When the metrics are input then development effort and success possibility of the project can be estimated by the model. The outputs of the model are important supporting information to help the sponsors and developers to make decisions about the next process.

4.2 The Measurement Model for Safety Analyses

Safety is a critical to many high confidence systems of embedded systems, especially for DoD systems. Software safety focuses on the failures of the system as they relate to hazardous events. A system is considered as "safe" if the probability of a hazardous failure has been reduced to some defined acceptable level. Safety is not a Boolean value of purely safe or unsafe, but a variable that ranges from completely unsafe towards safe [31, 32]. We developed a formal Instantiated Activity Model (IAM) and a metric to measure the probability that a hazardous event will occur and the severity of that hazardous event [30].

Instantiated Activity Model (IAM)

The IAM is a typical Input-Process-Output (IPO) block schema dealing with a set of related activities such as, input, process, output, failure, malfunction, etc. Figure 2 gives an example of an IAM. This is a typical IPO block with possible failure attached to the activities. For instance, Input I_1 with potential failure F_1, through successive activities Process P_1 with potential failure F_2 and Output O_1 with potential failure

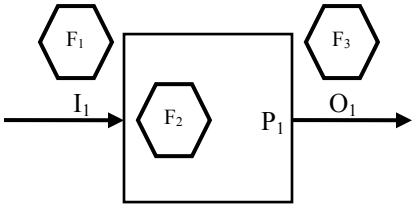

Figure 2. An Instantiated Activity Model

F_3 would result in a failure leading to a malfunction. The IAM reveals the relationship between essential IPO activities, the potential failures, and a hazardous situation or malfunction so we can establish a metric base for the safety analysis and risk assessment.

Hazard Probability of the IAM

The IAM is the key that supports formal approach for system safety analysis and risk assessment. This is based on the probability that a hazardous event will occur and the severity of that hazardous event (i.e., the consequences). Through the combination of these two elements, we can derive the hazard probability for the system as follows:

$$P_H(g) = \sum_i P_f(F_i, g) * P_e(A_i) * P_e(A_i\{DA_i\})$$

where $P_f(F_i, g)$ stands for probability of activity failure at degree *g*, *g* is the failure severity degree, $P_e(A_i)$ stands for probability of activity execution, $P_e(A_i\{DA_i\})$ stands for the probability of execution of A_i and $\{DA_i\}$, $\{DA_i\}$ stands for the dependent activities caused by activity A_i, A_i is the i[th] element of *A*, $A = I * O \cup R$, $I = \{I_1, I_2, I_3, \cdots | \text{all possible input activities}\}$, $O = \{O_1, O_2, O_3, \cdots | \text{all possible output activities}\}$, $S = \{R_1, R_2, R_3, \cdots | \text{all possible process activities}\}$.

The goal of making the IAM measurable on probability of failure is to identify potential hazards before the start of development, balancing development against effect. This method is especially effective for systems of systems. We can assume that each component system may have a myriad of different process flows that ultimately may result in a malfunction. We determine single failure probabilities using appropriate methods, as well as the determination of applicable process execution and related execution probabilities. It is possible to derive the probability that the whole system with execute a malfunction.

The risk exposure is the hazard probability times the cost of hazard occurrence.

5. Automated Software Generation based on Computational Models

DDAD integrates key processes in the software life cycle by the documentation management system (DMS). Models, activities, prototypes, simulations involved in these processes will be stored and manipulated in DMS. Supported by DMS, automated program generation can be realized based on a well-defined computational model and series of relevant techniques. A computational model was developed to describe the emergent properties, the interactions between component systems, and constraints associated with both functional and non-functional properties of a SoES [20]. A SoES ζ is modeled as follows:

$$\zeta = (S, E, C, D, F_1, F_2)$$

S is the component system set, $S = \{s_i \mid i \in [1, n]\}$, s_i denotes the component system constituting SoES (n is the number of component systems in the whole SoES); $E = \{e_{jk} \mid j, k \in [1, n]\}$ denotes the interaction sets between component systems, e_{jk} denotes the set of interactions from component system s_j to component system s_k; $C = \{c_i \mid i \in [1, n]\}$ denotes constraint sets on how the component systems are used in the given environment. c_i is a set of constraints on s_i. $D = \{d_{jk} \mid j, k \in [1, n]\}$ denotes constraint sets on interactions between component systems, d_{jk} is a set of constraints applied to interactions in e_{jk}.

Constraint sets *C* and *D* include the constraints for the design phase. They are refined from emergent properties *G* and high confidence constraints *H* of a SoES,

$$C = F_1(G, H); \quad D = F_2(G, H),$$

where F_1 and F_2 are two maps that map emergent properties and high confidence measures into local constraint sets on component systems and local constraint sets on interactions between component systems respectively. The mappings specify what must be assessed to ensure that the SoES satisfies its requirement

with high confidence, if it has already been certified that the individual s_i meet their requirements with high confidence. The constraint sets also represent a design for the systems integration, which will be realized by wrappers around the s_i.

Based on this model, a prototype system can be established to validate the requirements for a SoES. Well-formulated prototyping documentation can be used to promote system transition by extracting compositional architecture and evolving components. We found a way to build an explicit architecture for a prototyping system so that the product system can evolve through a transitional procedure [29]. The compatible composition model allows both explicit architecting and componential evolving by incorporating computer-aided prototyping techniques into a transitional process. Additionally, we introduced an object-oriented model for interoperability via wrapper-based translation [28]. This model performs transition from a computational phase, through a compositional phase, to a componential phase. During the transitional process, documentation passes throughout the development process. These results support automated software generation.

6. Development Knowledge Sharing Based on Ontologies

Collaboration capability between stakeholders is another important feature of DDAD. Effective sharing of information and interoperation of development artifacts are vital to collaborative software development, e.g. development of SoES. Ontology is now widely used for realizing knowledge sharing between organizations and/or individuals who have different culture backgrounds. Ontology is the term used to refer to the shared understanding of some domain of interest that may be used as a unifying framework to solve problems in that domain [24]. An ontology is a set of definitions of content-specific knowledge representation primitives: classes, relations, functions, and object constants. We have studied how to establish the software development tool ontology to improve interoperability in heterogeneous software development [13]. The methodology for constructing an ontology consists of 6 steps: (1) Identifying the purpose and scope of the ontology; (2) Feature modeling; (3) Establishing commonalities; (4) Determining tool ontologies; (5) Representation of the domain; (6) Documenting the ontology. The ontologies are important parts of the documentation repository to support collaboration between stakeholders.

(1) *Identifying the purpose and scope of the ontology*. One of the most important steps in constructing an ontology is to make an early decision about the purpose of the ontology. This purpose provides a controlling perspective on the terms, attributes of terms, and relationships captured in the ontology. The scope of the ontology provides a guide to the depth and breadth of the intended ontology, consistent with the purpose.

(2) *Feature modeling*. This step is to perform a domain analysis of software development tools by constructing and then considering the feature models of tools. Feature modeling is a method used to help define software product lines and system families, to identify and manage commonalities and variabilities between products and systems [14]. Feature models represent an explicit model of a device or system by summarizing the features and the variation points of the device/system. A feature model for software system captures the reusability and configurability aspects of reusable software. As an example, Figure 3 illustrates a feature model of a how PSDL timing constraints are implemented in CAPS.

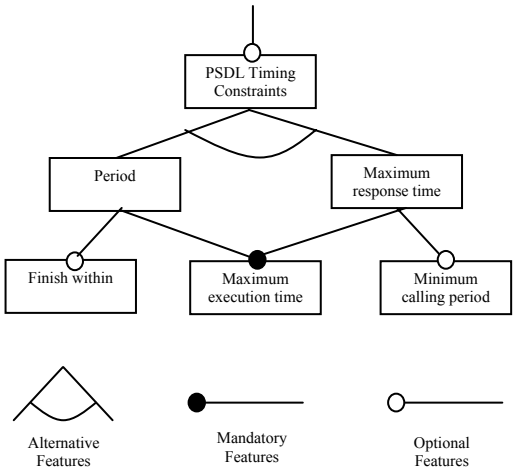

Figure 3. Feature Model of the PSDL Timing Constraints of CAPS

(3) *Establishing Commonalities*. This step is to isolate and annotate the commonalities that exist between the feature models. These common features then form the basis for the basic ontology terminology of the software development tool federation. The approach in this step is to reason about the feature diagrams, develop lists of potential terms from the feature diagrams, identify common terms between the lists, and then construct affinity diagrams of these common terms. Affinity diagrams are hierarchical Venn diagrams that provide groupings of related terms. The groupings of terms in the affinity diagrams then provide the basis for the hierarchy of terms in the software development tool ontology.

(4) *Determining tool Ontologies*. This step is the construction of the detailed ontologies of the tools to be used. In the case of tool ontologies, the detail needed for interoperability is dictated by the detail available through the API or source code (which ever is available) of the tool. Therefore, the ontology is derived from a selected set of classes and public methods related to the artifacts that are to be transmitted to (or received from) other software tools.

(5) *Representation of the Domain*. The fifth step requires that the relationships between all ontologies be identified and annotated. UML can be used to represent inter-relationship of ontologies. Such representations then make it possible to construct a set of all federation entities in the domain. When augmented with attribute computation rules, this representation can be made effective.

(6) *Documenting the Ontology*. The final step is to document the ontology. All assumptions about the domain and information about the meta-data used to describe the ontology should be annotated in the documentation repository in the form of template-based knowledge representation.

7. Methods and Models for Interoperability

We developed an Object Oriented Model for Interoperability (OOMI) to capture the information required for resolving the representational differences that exist in autonomously developed systems [33, 34]. Defining the interoperation between systems in terms of an object model provides a foundation for easy extension as new systems are added to an existing federation of systems.

The real-world entities and behavior information shared among a federation of interoperating systems are modeled in the OOMI using the concept of a *Federation Entity (FE)*. For each FE, one or more *Federation Entity Views (FEVs)* are used to distinguish the differences in the state and behavior information used for representing the same real-world entity on different systems (Figure 3).

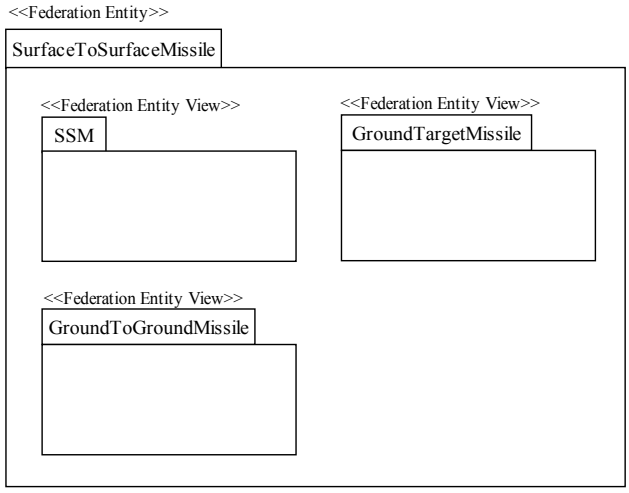

Figure 3. Defining Federation Entity (FE) and Federation Entity Views (FEVs) for Real-World Entity

It is expected that for a federation of heterogeneous systems, a number of real-world entities will be involved in the interoperation between systems. Under the OOMI, the collection of real-world entities used to define the interoperation of a specified federation of systems is termed a *Federation Interoperability Object Model (FIOM)* (Figure 4).

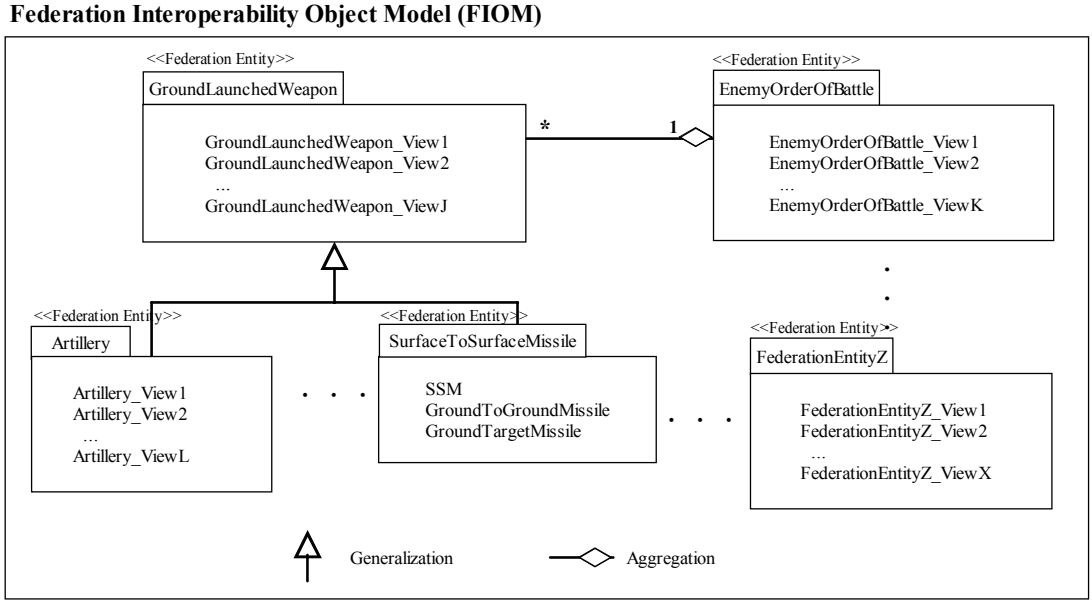

Figure 4. Federation Interoperability Object Model (FIOM) Representation

We also provided a Translation Generator for the Interoperability Engineers (IE) to define correspondences between the federation and component models' attributes and operations and generate the translation code skeletons, which can be modified to add functional or other transformations as necessary to resolve representational differences via the OOMI IDE facilities. The resultant wrapper-based Translator uses the FIOM, which the IE constructed using the OOMI IDE, to reconcile differences in real-world entity view and representation among component systems of a federation at run-time.

The initial use of the model is targeted for integration of legacy systems. Although these legacy systems generally have not been developed using object-oriented paradigm, an OOMI can easily be constructed from the external interfaces defined for most legacy systems (whether object-oriented or not).

We investigated formal models and mechanisms for describing the QoS attributes and techniques to assure the specified QoS. We developed a framework that allows an interoperation of heterogeneous and distributed software components. The framework incorporates (1) a meta-component model that describes the components, their services and service guarantees, and the infrastructure for integrating different component models and sustaining cooperation among heterogeneous components, (2) formal specification of components based on a two-level grammar, (3) validation and assurance of QoS based on event trace, and (4) generative rules for assembling a set of components out of available choices. We developed a Quality of Service behavior model based on the event trace analysis. The event trace approach allows us to directly examine specific quality of service actions that take place during program operation. In addition, we developed techniques to provide decision support for optimizing distributed object servers utilization, as well as the use software decoys to improve the security of systems of embedded systems [35, 36].

8. Applications of DDAD

8.1 Joint Tactical Radio System (JTRS)

The Joint Tactical Radio System (JTRS) is a revolutionary communications system that will be the foundation for all future Department of Defense tactical radios. JTRS will provide America's warfighters with state-of-the-art, software re-programmable, multi-band/multi-channel, network-capable systems that offer an interoperable, flexible and adaptable network for simultaneous voice, data and video communication [10]. It will create seamless interoperability and linkage among all military's air, land and sea legacy radio networks. Varied configurations of the system will advance communications mission requirements. The JTRS attribute of extendibility supports incorporating changes that are typical of many emerging requirements. In general, new requirements will be satisfied without hardware change provided the new waveform fits within certain bandwidth, data rate and transmission frequency bounds.

JTRS is a typical real-time, embedded, distributed, heterogeneous, and software-intensive system. The software implementation in JTRS should be able to dynamically adapt to the radio environment in which it is located at different times. A powerful documentation management system is needed for the JTRS program. Development of JTRS is complex and long-term. JTRS will be developed in several stages: Cluster 1 represents the first segment of the joint tactical radio system. The planned Clusters 2, 3, and 4 will address the handheld, maritime, and airborne needs. A team led by Boeing has been selected to begin building common tactical radios. The Boeing team is comprised of many sub-teams that take charge of different tasks [5].

A knowledge sharing and management environment can be constructed based on the idea of the documentation management system (DMS). This environment will support the decision coordination and cooperation between development teams. The documentation repository can be used in not only software development but also system and hardware development of JTRS as long as the related knowledge is appropriately represented in the form of template-based knowledge representation. The maintainability, traceability, consistency, understandability of documentation repository and the ability of quickly tracking and responding changes in requirements will increase the efficiency and decrease the risk of the development of JTRS. This application requires attention to the finer points of developing a distributed implementation of the DMS.

8.2 Ballistic Missile Defense Simulation Systems

The evolving ballistic missile defense problem must be solved to support a long-term strategy that calls for an integrated and adaptable "system of systems" to defend U.S. territory, forces, allies, and other interests worth protecting [2]. Credible Department of Defense models and simulations (M&S) of ballistic missile defense systems are expected by National- and Department-level decision-makers [6]. Many of these large-scale, software-intensive simulation systems were autonomously developed over time, and subject to varying degrees of funding, maintenance, and life-cycle management practices, resulting in heterogeneous model representations and data. Systemic problems with distributed interoperability of these non-trivial simulations in federations' persist, and current techniques, procedures, and tools have not achieved the desired results. Establishing credibility in DoD simulations involves many disciplines and knowledge areas including software engineering, processes, quality, product management and architecture. The Department's complex organizational dynamics, and complicated acquisition procedures also impact the level of M&S credibility, at times adversely.

There are two ways to apply the idea of DDAD to ballistic missile defense simulation systems. One way is to use DDAD directly in the development of simulation software that is credible. The other way is to apply the main idea of DDAD in simulations. A documentation management system for simulations can be built. This will enable all information involved in simulations to be well organized and manipulated so that the simulation processes are transparent, traceable and maintainable. Credibility of the simulation results will therefore be improved.

8.3 Joint Forces Program

Joint forces are now more important than ever because in today's world the traditional distinctions between maritime, land and air theatres of operations have become less relevant. By operating as a single, united force, the Navy, Army and RAF can produce a bigger punch, maximizing operational effectiveness and increasing the chance of success [7]. Interoperability requirements are critical to joint force programs. Since interoperability requirements are dynamic, and often poorly understood before systems are put to use in the field, the requirements and acquisition communities must have a flexible and powerful method to communicate in order to overcome these challenges.

Based on the idea of DDAD, we have proposed a unified repository of architectural data, with the ability to be viewed in several forms (i.e. with the ability to create multiple architectural views), each tailored to the needs of different stakeholders [12]. The power of this methodology is that it provides a mechanism by which functional and interoperability requirements are captured, defined, and levied on systems based on how they will be employed. This is a dynamic process, which can accept changes to requirements, system environments, and domains; and which supports time-phasing, spiral development, assessment of requirements vs. capabilities and operational vs. system needs.

9. Conclusions

This paper explores a new view of documentation that can better serve development of systems of embedded systems. The different views provided by the DDAD approach give project managers, developers, sponsors, maintainers and end-users the ability to express their opinions or propose requirements changes if needed by adding related documents via a user-friendly interface. This information will be recorded in a form that can be manipulated, automatically analyzed and made available throughout the rest of the development process. DDAD will track these changes and help to ensure that information will not be corrupted in transformation from one phase to another. DDAD provides a method that encourages stakeholder involvement while updating the requirements and consistently providing this information for later use. DDAD also supports automated software generation by using a computational model, rapid prototyping and other related techniques. This is helpful to achieve a good tradeoff between stakeholder interaction and process automation. DDAD also provide a method to monitor and respond to frequent changes in requirements. Consequently, agility of the development will be greatly increased.

By using the DDAD approach in every phase of development, even the automated processes, it should become practically feasible to record, compile and present information to different stakeholders and tools in a clear, understandable way at a level of complexity required to meet the stakeholders' needs. By having these different views available at various stages of development, stakeholders will be able to effectively monitor the development process and communicate with each other. This improved transparency provides valuable information needed for quality control and overall process improvement.

Software development processes from one phase to another are embodied as capture of relevant information (e.g., design specification, quantifiable attributes), definition of document information models and view presentation models, simulation of semantic behavior (e.g., executable specification), and transformation of documents exploited by various phases. With insight into the future development of documentation, the documentation repository will support transformation from high-level description (in some specification languages) to low-level description (in some programming languages) with mapping between those descriptions.

DDAD also provides comprehensive support for software maintenance and evolution. In DDAD, all the activities and information used by the development processes are accurately recorded and organized in a well-formulated documentation system that drives the system development and build processes. This will ensure overall system properties are precisely documented and consistently updated and transferred throughout successive phases and available after system release. The documentation will retain sufficient detail to provide a sound basis for fault tracing, bug repairing and overall system improvement. DDAD will keep track of system configuration, document dependencies and system status and enable the software to respond to future changes in requirements thereby supporting maintenance and evolution of the system.

Keeping track of accurate dependency information is critical for automatically locating the relevant parts of a maze of documents for resolving a given system evolution issue.

From the viewpoint of long-term system construction, technologies for computer-aided documentation repositories will drive the form of documentation standard needed for more effective regulatory management. Much of the presented infrastructure can be generalized from software development to the entire systems engineering and certification process.

DDAD will be a promising methodology to build a high confidence system of embedded systems. Three potential applications were presented in the paper, but the methodology and idea of DDAD can be used in many more industrial domains.

Reference

[1] B. Boehm, "Software Risk Management: Overview and Recent Developments", 17th International Forum on COCOMO and Software Cost Modeling, Los Angeles, CA, October 22-25, 2002, http://sunset.usc.edu/events/2002/cocomo17.

[2] D. C. Gompert, J. A. Isaacson, "Planning a Ballistic Missile Defense System of Systems", http://www.rand.org/publications/IP/IP181/.

[3] E. Hall, Managing Risk. *Methods for Software Systems Development.* Addison Wesley, 1997.

[4] J. M. Shridhar, S. Ravi, "Virtual Manufacturing: An Important Aspect of Collaborative Product Commerce", *Journal of Advanced Manufacturing Systems*, Vol. 1, No. 1, 2002, pp. 113-119.

[5] http://www.boeing.com/news/releases/2002.

[6] http://www.sc.army.mil/.

[7] http://www.mod.uk/aboutus/factfiles/jointforces.htm.

[8] http://www.extremeprogramming.org.

[9] http://www.dsdm.org.

[10] J. H. Reed, *Software Radio: A Modern Approach to Radio Engineering*, Prentice Hall, 2002.

[11] J. Highsmith, "Agile Software Development: A Review of Agile Methodologies," http://www.cutter.com/workshops, December, 2002.

[12] J. L. Parenti, "Engineering Software for Interoperability Use of Enterprise Architecture Techniques", *Master Thesis*, Naval Postgraduate School, March 2003.

[13] J. Puett, "Holistic Framework for Establishing Interoperability of Heterogeneous Software Development Tools", *Ph.D Dissertation* (advisor: Luqi), Naval Postgraduate School, June, 2003.

[14] K. Czarnecki, U. Eisenecker, *Generative Programming Methods, Tools, and Application*s, Addison-Wesley, 2000.

[15] L. Putnam, and W. Myers, *Industrial Strength Software*: *Effective Management Using Measurement*. IEEE Computer Society Press, 1997.

[16] V. Berzins, L. Qiao, Luqi, "Information Consistency Checking in Documentation Driven Development for Complex Embedded Systems", submitted to Monterey Workshop 2003, Chicago, USA, September 24-26, 2003.

[17] Luqi, M. Ketabchi, "A Computer-Aided Prototyping System", *IEEE Software*, March, 1988, pp. 66-72.

[18] Luqi, R. Steigerwald, et al, "CAPS as a Requirement Engineering Tool". in *Proceedings of Tri-Ada'91 International Conference*, San Jose, USA, Oct 22-25, 1991, pp. 75-83.

[19] Luqi, V. Berzins, R. Yeh, "A prototyping language for real time software", *IEEE Transactions on Software Engineering*, Vol 14, No 10, 1988, pp. 1409-1423.

[20] Luqi, Y. Qiao, L. Zhang, "Computational Model for High-confidence Embedded System Development", Monterey Workshop --- Radical Innovations of Software and Systems Engineering in the Future, October, 7-11, 2002, pp. 265-303.

[21] M. Lyu, *Software Reliability Engineering.* IEEE Computer Society Press. 1995.

[22] M. Murrah, "Enhancements and Extensions of Formal Models for Risk Assessment in Software Projects", *Ph.D Dissertation* (advisor: Luqi), Naval Postgraduate School, September, 2002.

[23] M. Saboe, "A Software Technology Transition Entropy Based Engineering Model", *Ph.D Dissertation* (advisor: Luqi), Naval Postgraduate School, March, 2002.

[24] M. Uschold, M. Gruninger, "Ontologies: Principles, Methods and Applications," *Knowledge Engineering Revie*w, Vol. 11, No. 2, June 1996.
[25] N. Johnson, and S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions*. Vol. 1. Wiley & Sons, 1994.
[26] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta, "Agile Software Development Methods-Review and Analysis", *Technical Report*, ESPOO 2002.
[27] P. M. Nelson, "A Requirements Specification of Modifications to the Functional Description of the Mission Space Resource Center", *Master Thesis*, Naval Postgraduate School, June 2001.
[28] P. Young, V. Berzins, J. Ge and Luqi, "Use of Object-Oriented Model for Interoperability in Wrapper-Based Translator for Resolving Representational Differences between Heterogeneous Systems", *Monterey Workshop 2001 on Engineering Automation for Software Intensive System*, Monterey, CA, 2001, pp. 170-177.
[29] X. Liang, J. Puett and Luqi, "Perspective-based Architectural Approach for Dependable Systems", *Proc. of ICSE 2003 Workshop on Software Architectures for Depenable Systems*, Portland, OR, USA, May 3, 2003, pp. 1-6.
[30] Luqi, X. Liang, M. Brown, C. Williamson, "Formal Approach for Software Safety analysis & Risk Assessment via an Instantiated Activity Model", to appear in the 21th International System Safety Conference, August 4-8, 2003, Ottawa, Ontario, Canada.
[31] National Aeronautics and Space Administration, *NASA Œ STD Œ 8719.13A, Software Safet*y, NASA Technical Standard, September 15, 1997.
[32] United Kingdom Ministry of Defense, *Ship Safety Management System™s Handboo*k, JSP 430, UK.
[33] P. Young, V. Berzins, J. Ge and Luqi, "Using an Object Oriented Model for Resolving Representational Differences between Heterogeneous Systems", *Proceedings of 17th ACM Symposium on Applied Computing (SAC)*, Madrid, Spain, 10-14 March 2002, pp. 976 - 983.
[34] P. Young, "Integration of Heterogeneous Software Systems through Computer-Aided Resolution of Data Representation Differences", *Ph.D. Dissertation* (Advisor: Luqi), Naval Postgraduate School, Monterey, CA, March 2002.
[35] W. Zhao, B. Bryant, R. Raje, M. Auguston, A. Olson and C. Burt, "A Unified Approach to Component Assembly Based on Generative Programming", *Proceedings of 2002 Workshop on Generative Programming (GP 2002)*, Austin, Texas, April 2002, pp.195-199.
[36] J. Drummond, Luqi, W. Kemple, M. Auguston and N. Chaki. "Quality of Service Behavioral Model from Event Trace Analysis." *Proceedings of the 7th international Command and Control Research and Technology Symposium (CCRTS 2002),* Quebec City, Quebec, 16-20 September 2002.
[37] K. Beck et al., "Manifesto for Agile Software Development", www.agilemenifesto.org, February 2001.
[38] K. Back, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.
[39] T. DeMarco, B. Boehm, "The Agile Methods Fray", IEEE Computer, Vol. 36, No. 6, 2003, pp. 90-92.
[40] Luqi, Z. Guan, "A Software Engineering Tools for Requirement Document based Prototyping", *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Infromatics*, Orlando, Florida, USA, July 27 - 30, 2003, Volume VI, pp.237-243.