Risk Assessment of Software-System Specifications

Hany H. Ammar, *Member, IEEE*, Tooraj Nikzadeh, and Joanne Bechta Dugan, *Fellow, IEEE*

***Abstract*—Summary & Conclusions—This paper presents a methodology and an example of risk assessment of functional-requirement specifications for complex real-time software systems. A heuristic risk-assessment technique based on CPN (colored Petri-net) models is presented. This technique is used to classify software functional-requirement specification components according to their relative importance in terms of such factors as severity and complexity. A dynamic complexity measure, based on concurrence in the functional requirements, is introduced. This technique is applied on the Earth Operation Commanding Center (EOC_COMMANDING), a large component of the NASA Earth Observing System (EOS) project. Two specification models of the system are considered. Results of applying this technique to both models are presented.**

The risk assessment methodology in this paper suggests the following conclusions:

- **Risk assessment at the functional-requirement specification phase can be used to classify functional requirements in terms of their complexity & severity. The methodology identifies high-risk functional specification components that require appreciable development & verification resources during design, implementation, and testing.**
- **Dynamic Complexity metrics and the concurrence metric (introduced in this paper) can important in assessing the risk factors based on the complexity of functional specifications.**
- **The Concurrence complexity metric (introduced in this paper) is an important aspect of dynamic complexity.**
- **CPN models can be used to build an executable specification of the system, which helps the analyst not only to acquire deep understanding of the system but also to study the dynamic behavior of the system by simulating the model.**

Future research in early risk assessment and complexity analysis could focus on:

1) **Software Architectures based on Object Technology: The technique in this paper, with some modifications on complexity analysis and severity analysis, applies to the design methods and software architectures based on object technology. Further research is required to establish the complexity metrics for object-based systems.**
2) **SRE (Software Reliability Engineering): One of the main tasks in SRE is designing the operational profiles. Operational profiles are built according to the user profile and the understanding of the system analyst/designer. These profiles can be used for estimating the system reliability at the early phases of development. Results obtained from this analysis can be incorporated into SRE for conducting reliability analysis at the analysis/design phases, based on dynamic simulation. More research is needed to establish a method for incorporating the risk assessment method within the SRE process.**

***Index Terms*—Colored Petri-net, risk assessment, severity measure, software complexity metric, software specification.**

Manuscript received September 4, 1998; revised March 9, 2000. This work was funded by a research grant from the U.S. NASA Goddard under Contract NAG 5-2129 to West Virginia University.

Responsible Editor: J.D. Healy

H. H. Ammar is with the Dept. of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506 USA (e-mail: HAmmar@wvu.edu).

T. Nikzadeh is with the 24/7 Media/IMAKE Software and Services, Bethesda, MD 20817 USA (e-mail: Tooraj@imake.com).

J. B. Dugan is with the Dept. of Electrical & Computer Engineering, Univ. of Virginia, Charlottesville, VA 22904-4743 USA (e-mail: JBD@virginia.edu).

Publisher Item Identifier S 0018-9529(01)09555-0.

ACRONYMS[1]

AOI	accept operator input commands
BSRC	build spacecraft real-time command
CAV	command authority violation
CPN	colored PN
cpx	complexity
CTN	count transmission number
EDOS	EOS data & operations system
EOC-CONT	EOS commanding controller
EOS	Earth observing system
EOSDIS	EOS data & information system
EVAL	evaluate spacecraft command status
FCL	fetch command load
FMEA	failure modes and effects analysis
FOS	flight operation segment
hrf	heuristic risk factor
ICC	instrument control center
MRG	merge command
PCA	principal component analysis
PN	Petri net
RCV	receive command status data
SART	structured analysis for real-time systems
TRAN	transmit command
VC	validate commands
VRFY	verify command

NOTATION

c_k	operational complexity factor for component k
C_p	structure complexity using design structure metrics
$C_{i;p}$	internal complexity measure of the module p
C_t	system complexity
CCF_k	concurrence complexity of component k
ccpx_k	concurrence complexity factor of component k
CCPX	system concurrence complexity
Cpx_k	normalized complexity metric of component k
$\mathbf{d}_k$	vector of orthogonal domain metrics associated with component k
D_t	data complexity
e	number of edges in a control flow graph
F_k	number of distinct fired transitions of component k throughout the whole simulation time
FAN-IN	number of modules that call a given module

[1]The singular & plural of an acronym are always spelled the same.

FAN-OUT number of modules called by a given module
fcpx_k functional complexity of component k
FCPX system functional complexity
g set of guards on the input arcs of a transition in a CPN model
G transition guard
HC_p information flow metric
Hrf_k hrf of component/module k
HRF system hrf
M number of input places of a transition in a CPN model
n number of nodes in a control flow graph
N number of output places of a transition in a CPN model
ocpx_k operational complexity of component k
OCPX system operational complexity
p number of principal components selected
p_k Pr{component k in a set of n components is in execution at any arbitrary time}
S_t structural complexity
scpx_k static complexity of component k
SCPX system static complexity
sscpx_k relative scpx_k
Svrty_k severity level of component k
T transformation matrix whose elements $t_{i,j}$ give the coefficient, or weight, of complexity metric i $(1 \leq i \leq 3)$ for domain metric j $(1 \leq j \leq p)$
VG cyclomatic complexity
VG_c VG of the code segment of a transition
VG_k VG of the total flow graph representing a nonprimitive component k
VG_t VG of a transition
Λ^* vector of eigenvalues associated with the selected domains

I. Introduction

THE objective of risk assessment in software systems is to identify the functional requirements which pose the greatest risk should errors occur. Using classic fault avoidance, fault removal, or fault tolerance techniques including redesign, formal methods, extensive testing, and redundancy can then mitigate the risk.

This paper classifies functional requirements in terms of complexity & severity. Complexity is risky in that complex systems are difficult to design, implement, and verify. Severity is a measure of the magnitude of the consequences, should an error be produced. Potential errors that can lead to severe consequences are targeted for special handling. The hrf is defined as a way to combine complexity & severity measures for component & system risk assessment.

The heuristic risk assessment method in this paper is based on CPN models of software requirement specifications. Several research studies appeared in the literature on using PN for software system modeling & analysis. Reference [14] used timed PN to evaluate and optimize the behavior of systems, and introduced an approach to transform requirements-driven PN models into logic programs containing the static structure and the dynamic behavior of the PN models. Reference [21] used timed PN for safety analysis of software systems. Reference [36] proposed an approach to Object-Oriented Software Design (assuming potential concurrence) using CPN. It argues that CPN conceptual mechanisms not only can be applied effectively for software modeling purposes, but also one can use the simulation facilities of the Design/CPN software package [40] to verify important system properties, to make design/implementation decisions, and to debug the design.

The hrf for requirements specifications defined in this paper is based on dynamic complexity and severity measures. This is different from previous work in the literature that was targeted to complexity analysis at the detailed design & implementation phases. Some of these studies have demonstrated that there exists high *s*-correlation between design and source-code complexity on one hand and quality factors, such as maintainability or reliability, on the other hand [2], [3], [4]. Reference [17] applied complexity metrics during the development of large telecommunication software to identify high-risk components and to tailor reliability-growth models. It used complexity-based metrics to predict criticality of the modules in the system. Doing so, it could identify the critical components and make predictions on the failure rate as early as possible in the software life cycle. For the complete approach of criticality prediction, recent data from the development of a switching system with about 2 MLOC was provided. The switching system is currently operational, thus allowing for validation & tuning of the prediction model.

Some predictive models incorporate a functional relationship of program error measures with software complexity metrics [18]. Software complexity metrics are also used for test-design and test-execution suites [19]. Complexity analysis & measurement benefit from applying principal component analysis, a proven mathematical technique, to the complexity metrics. Reference [20] used this technique for early prediction of software quality of a large telecommunication system to identify fault-prone modules prior to testing.

This paper bases complexity measurement of the software functional requirements on CPN specification models [1], [39]. Dynamic complexity (functional, operational, concurrence) measures are obtained from the analysis of the dynamic behavior of the system by simulating the CPN model.

Section II briefly outlines the background on CPN modeling and software complexity measures.

Section III describes the CPN specification models of the Commanding Component of the NASA EOS.

Section IV presents the methodology of risk assessment.

Section V presents the results of applying the risk assessment methodology on the Commanding component specification models.

II. Background

A. CPN

A simple CPN Net is composed of the following graphical elements [40]:

1) Places (represented by circles): locations for holding data.

2) Transitions (represented by rectangles): activities that transform data.
3) Arcs (represented by arrows) connect places with transitions, to specify data flow paths.
4) Arc Inscriptions: Input arc inscriptions specify the data that must exist for an activity to occur; Output arc inscriptions specify the data that will be produced if an activity occurs.
5) Guards define conditions that must be true for an activity to occur.

CPN use data types, data objects, and variables. CPN datatypes are referred to as color sets, and CPN data-objects are referred to as tokens. A guard is a Boolean expression associated with a transition or with input arcs. Transitions in a CPN could have code segments in which the exact transformation from input data to output data, using the CPN ML language, can be implemented [40].

The dynamic aspects of CPN models are denoted by markings, which are assignments of tokens to the places of a CPN model. A transition is enabled iff each of its input places contain at least as many tokens as there are arcs from that place to the transition. When a transition is enabled, it can fire enabling tokens to be removed from the input places. Tokens are deposited in each of the output places according to the code written in the code segment. At any given time instance, the 'distribution of tokens on places' defines the current state of the modeled system.

By adding g to the input arcs and/or by adding a G, one can control the enabling conditions of the transition. Therefore; the dynamics of firing a transition in a typical CPN model are summarized in the procedure:

```
Fire_Transition( ){
 —check for enabling conditions at the input arcs guards g
 if g then
   —check for enabling condition at the transition guard G
   if G then
     execute code c( ); the code attached to the transition,
       if any.
     deposit tokens to the output places;
     end_if;
   end_if;
end_Fire_Transition
```

B. Software Complexity Metrics

Researchers have proposed many complexity metrics & models over the years. In 1977, [6] established a software complexity metrics based on the primitive measures such as the number of distinct operators & operands, and the total number of operators and operands in the program. It developed a system of equations expressing some measures of software quality.

In 1976, [8] introduced a measurement of cyclomatic complexity as an indicator of testability & maintainability of a program. Cyclomatic complexity is the classical graph theory cyclomatic-complexity that shows the number of regions in a flow graph:

$$\mathrm{VG} = e - n + 1. \tag{1}$$

These complexity metrics, however, measure implicitly the complexity of a module as a separate entity. To take into account the interactions among system modules, several structure metrics have been introduced. McClure introduced invocation complexity (1978), system partitioning measures were presented by Belady & Evangelisti (1981), information flow metrics were proposed by Henry Kafura (1981). In 1980 [16] used stability measures, and defined structure complexity using design structure metrics as:

$$C_p = (\text{FAN-IN} \cdot \text{FAN-OUT})^2. \tag{2}$$

Selig defined a hybrid complexity to incorporate the information-flow metric in 1990 as:

$$\mathrm{HC}_p = C_{i,p} \cdot (\text{FAN-IN} \cdot \text{FAN-OUT})^2. \tag{3}$$

Reference [16] developed a C_t model which includes S_t and D_t:

$$\begin{aligned} C_t &= S_t + D_t; \\ S_t &\equiv \sum_k \frac{(\text{FAN-OUT of module } k)^2}{\eta}, \\ D_t &\equiv \sum_k \frac{D_k}{\eta}, \\ D_k &\equiv \frac{\text{I/O variables in module } k}{\text{FAN-OUT of module } (k+1)} \\ \eta &\equiv \text{ number of modules} \end{aligned} \tag{4}$$

Most of the complexity metrics are based on design specification.

III. EOS Specification Models

The NASA EOS, currently under development, is a large-scale distributed information system. EOS is anticipated to be the centerpiece of the NASA "Mission to Planet Earth". NASA plans to launch a total of 6 Earth-looking scientific observatories accompanied by other observatories from the European Space Agency. The instrumentation will focus primarily on measurement designed to enhance the understanding of the Earth's climate and hydrologic systems and ecosystem dynamics.

The FOS is responsible for EOS mission operation. FOS exchange information with other EOS components to obtain the necessary data for the generation & uplink of commands. It interfaces with EDOS for command uplink, and receives the real-time command from, and displays the messages to, the operators. An operator can be a Project Scientist or his Designee, Operations Manager,Principal Investigator, Team Leader or User.

Based on functional requirement specifications, and scheduling scenarios, it was observed that the Commanding component is important in FOS. A model of the Commanding Subsystem was built based on the requirements specifications document of NASA. These are listed here according to the Functional and Performance Requirements Specification Document for the EOSDIS Core System [34], and the General Requirements Document of the FOS Requirements Specification (volume 1) [35].

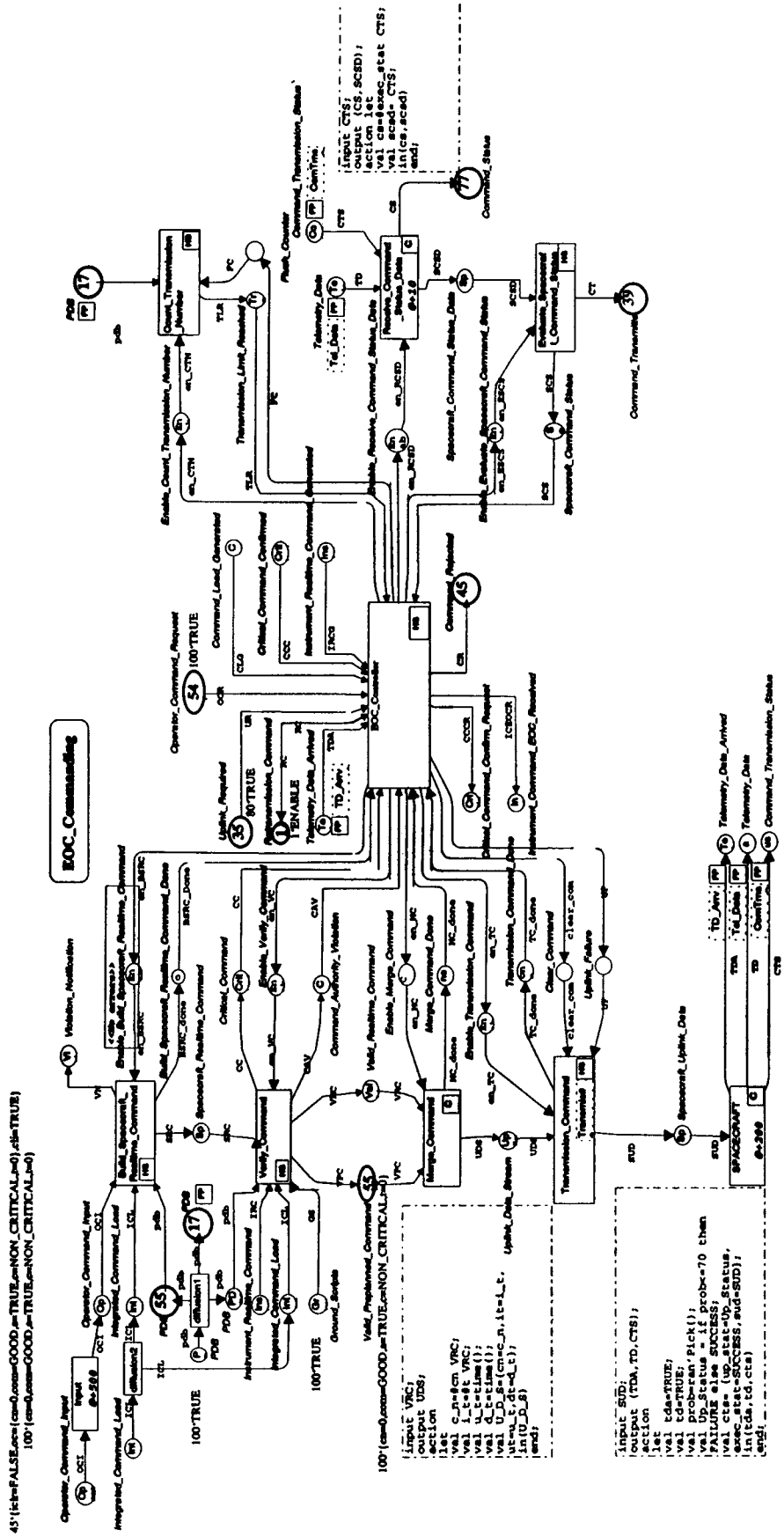


Fig. 1. The CPN model of the Commanding component of EOS.

- Generate and verify real-time commands. This is accomplished by the functions Build-Spacecraft-Realtime-Command and Verify-Command.
- Merge and uplink the preplanned and real-time commands to EDOS. The functions Merge-Command and Transmit-Command are responsible for this task.
- Receive and evaluate the command status. This is accomplished by the functions Evaluate-Spacecraft-Command-Status and Receive-Command-Status-Data [34].
- Automatic retransmission is also provided when an unsuccessful transmission occurs. This is managed with the help of function Count-Transmission-number [34].

In order to fulfill these 4 tasks, `Commanding` is decomposed into subfunctions:

- Build-Spacecraft-Realtime-Command (BSRC)
- Verify-Command (VRFY)
- Merge-Command (MRG)
- Transmit-Command (TRAN)
- Evaluate-Spacecraft-Command-Status (EVAL)
- Receive-Command-Status-Data (RCV)
- Count-Transmission-Number (CTN).

These subfunctions can be activated concurrently to build & uplink a stream of commands. Fig.1 shows the CPN model of commanding, and shows the data flow/control flow model of these 4 functions, where 'transitions' model functions and 'places' specify data-flow and control-flow specifications. This model is described briefly in Section III-A.

A. CPN Model

In Fig.1, BSRC generates the spacecraft realtime command based on the operator command input and the preplanned command script. The Verify-Command function checks the authorization level of a command and determines whether a specific command is critical, based on its definition [34]. The Merge-Command function merges the Valid-Preplanned-Commands and Valid-Realtime-Commands resulting from the Verify-Command function into one Uplink-Data-Stream.

The Transmission-Command function receives the Uplink-Data-Stream from Merge-Command and uplinks the commands. The CTN function counts the number of times a command was uplinked. When an Uplink-Failure occurs, the Retransmission-Command flag is checked. If the flag is set to Enable and the Transmission-Limit is not reached, then the Command is retransmitted. The operator can set the Transmission-Limit to a suitable number of retransmissions based on the uplink conditions.

The Evaluate-Spacecraft-Command-Status function verifies the successful receipt & execution of all commands by the spacecraft [34]. It evaluates the command uplink status and the command execution status. Both evaluations are based on the telemetry data. The subfunction Evaluate-Spacecraft-Command-Uplink-Status evaluates the spacecraft command-uplink status to verify the successful receipt, and the subfunction "Evaluate-Spacecraft-Command-Execution-Status" evaluates the execution status of the Commands by the spacecraft itself.

The Spacecraft function receives the Spacecraft-Uplink-Data and produces as output the Command-Transmission-Status and Telemetry-Data. The code segment for this transition consumes the tokens provided for the spacecraft through transmission, and produces Command-Transmission-Status, Telemetry-Data, and Telemetry-Data-Arrived token to be used by the Received-Command-Status-Data function. A probability distribution on the success or failure of the uplink-status of the command must be specified in the code segment. This distribution can be set according to the desired simulation scenario.

The EOC-CONT transition in Fig.1 produces the enabling control flow signals to these functions. This controller sub-model gives the control specifications for the Commanding component as a whole. It produces the tokens necessary for enabling the functions of the Commanding component and sends messages to the operator.

B. The Pipelined Specification vs the Sequential Specification Models

The flexibility of CPN notation to express the control flow of a software specification greatly reduces the amount of effort needed to design alternative specifications and to explore its behavior under specification changes.

The specification sub-model of the EOC_CONT in Fig.1 was developed first for a sequential processing of commands. The specification was then changed to allow for processing a stream of commands in a pipeline fashion. The change allows the enabling of transitions modeling the functions BSRC, VRFY, MRG, TRAN to be done concurrently.

In the sequential model, when VRFY processes the output of BSRC, all other processes are inactive and the next Operator_Command_Input is processed only after the current command is transmitted. In the pipelined specifications several commands are processed by the system, each in a specific stage of processing.

IV. METHODOLOGY

Traditionally, system reliability is measured either by the time between failures, or by the number of faults within a certain period of time. These two measures are, however, interchangeable. There is a *s*-correlation between the number of faults and the complexity of the system [5], [6]. The more complex a system is, the more faults the system is anticipated to have. However, to improve software quality management, we must be able to predict early on, in the development process, the components of the software system that are likely to have a high fault rate. Hence, complexity metrics are used in this study.

On the other hand, we must also identify the system components that require special development-resources due to their severity and/or criticality. There could be a relatively low-complex component (or subsystem) that has a safety role in the system. The effect(s) of failures in such component could be catastrophic. Therefore, risk-assessment considers also the severity associated with each component based on how its failure(s) affect the system operation & performance.

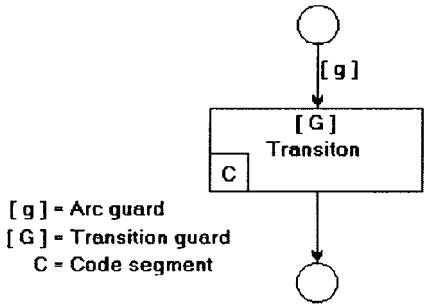


Fig. 2. A primitive transition with an arc guard, a transition guard, and a code segment.

A. Complexity

Characterizing the software quality in terms of a set of measurable attributes is the main objective in any software measurement process. Based on the observation that software complexity has a direct impact on its quality [6], we focus on complexity metrics. We anticipate a higher likelihood of failures for components with high complexity. The important fact is that measures of software complexity can be obtained very early in the software life cycle.

Several software complexity metrics are:

- Halstead metrics [15],
- lines of code (LOC),
- control flow graph,
- data flow graph,
- number of fault counts,
- number of change counts.

Linear combinations of these elements are measures for most of the existing software metrics. However, these measures are interrelated. To resolve this problem, we apply the statistical technique, PCA. PCA detects & analyzes the relationships among software metrics, and provides a new set of orthogonal variables, known as principal components, that conveys almost all of the information in the original set. The principal components are constructed to represent transformed scores on dimensions that are orthogonal [7].

1) Static Complexity (scpx, SCPX): The complexity model in this paper is based on the CPN model of the functional specification of the system. This complexity model includes the number of inputs to, and outputs, from a component and its McCabe cyclomatic complexity obtained from the control-flow graph. Based on the system CPN model, we use the metrics M, N, VG, for static complexity analysis.

VG is the number of *s*-independent paths in a strongly connected graph. McCabe developed this nonprimitive metric which measures some aspects of control-flow complexity [8]. This number can be calculated by constructing the control-flow graph in a computer program. In such a directed graph, nodes represent entry points, exit points, code segments, or decisions in the program; edges represent control flow. The transition in Fig.2 has the control-flow graph in Fig.3; see the Fire_Transition() procedure in Section II-A.

The VG of this transition is: VG $= 2$, from (1). The control-flow graph of the code segment is also considered in calculating VG. For that component, $\mathrm{VG}_t = \mathrm{VG} + \mathrm{VG}_c$.

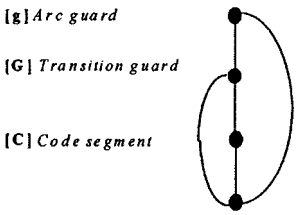


Fig. 3. Control flow graph of Fire_Transition().

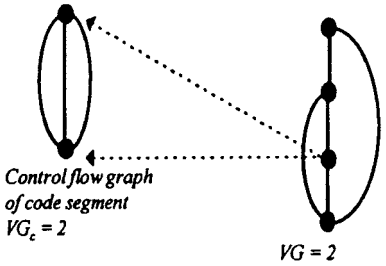


Fig. 4. VG for the transition and the code segment.

Fig.4 is a case where the code segment has a complexity of 2 which is then added to the complexity of the transition.

The system complexity in the model is the sum of the complexities of its subsystems/components. PCA is used to estimate the static complexity of the system components. PCA is applied on the three mentioned complexity metrics. The rational for using PCA is based on the notion that the metrics M, N, VG are *s*-correlated, which is because the inputs & outputs of a transition in the model are used to represent both data-flow and control-flow activities. VG captures the complexity of processing the inputs and of producing the outputs; hence it depends on M, N. The following paragraphs briefly & informally summarize the results of applying PCA; for more details, see [7], [33]. Reference [33] contains a detailed discussion on applying PCA to the EOS model in Section III.

A software quality model is a statistical relationship between independent variables (e.g., complexity metrics) and a dependent variable (e.g., software complexity measure). After calculating the model parameters, then calculate the value of the dependent variable, given the set of independent variables. We begin with the original data-set matrix. Each row in this matrix corresponds to a primitive component, and each column corresponds to the complexity measures of this component, M, N, or VG.

PCA transforms raw-data variables into new variables (principal components) that are *s*-uncorrelated [9]. When the underlying data-set is a software metric, then each principal component is a domain metric. The principal components represents the same data in a new coordinate system, in which transformation is chosen both to maximize the variability in each direction and to make the principal components uncorrelated [9]. The number of principal components in this example is at most 3. Although PCA is usually applied to problems that have many variables, we summarize the 3 variables (M, N, VG) with 1 variable (scpx) as shown in (5). The technique could be easily applied if there were many other complexity metrics besides the 3 used here.

Applying PCA (as in [6]), a T can be obtained such that its $t_{i,j}$ give the coefficient/weight of complexity metric i $(1 \leq i \leq 3)$ for domain metric j $(1 \leq j \leq p), p \leq 3$. The d_k is obtained by post-multiplying 'row k of the data-set matrix containing the complexity measures of component k' by T. d_k has a mean $= 0$ and a variance $= 1$, assuming T is a standardized matrix.

Once the domains are identified, one can compute the relative complexity metric for each program component k by forming the weighted sum of the domain metrics as follows:

$$\begin{aligned} \text{scpx}_k &= d_k \cdot \Lambda^{*'} \\ \Lambda^{*'} &\equiv \text{ transpose of } \Lambda^*. \end{aligned} \tag{5}$$

This relative complexity metric is the weighted sum of a set of uncorrelated attribute domain metrics. This metric is justified in [6].

Relative static complexity

A scaled version [10] of this metric, (6), is the relative static complexity metric. This metric is more easily interpreted [6]. For example, the scaled sscpx_k can be several standards of deviations above the mean relative complexity for all components in the system.

$$\begin{aligned} \text{Sscpx}_k &= \frac{\text{STDEV} \cdot \text{scpx}_k}{\sqrt{V(P)}} + \text{MEAN}; \\ V(P) &\equiv \text{sum of the square of the eigenvalues.} \end{aligned} \tag{6}$$

The scaled metric is distributed with a mean of MEAN and a standard deviation of STDEV (typical values for STDEV and MEAN are usually selected as $\text{STDEV} = 10$, and $\text{MEAN} = 50$).

SCPX is then the sum of the static complexities of all the components (modules) comprising the system:

$$\text{SCPX} = \sum_k \text{sscpx}_k \tag{7}$$

2) Dynamic Complexity (dcpx, DCPX): The scpx is a measure of the program at rest. However, when a program is running, the level of exposure of its components is a function of the execution environment (operational profile and the platform). Consequently, both the 'static complexity' and 'system's operational environment' influence the system reliability [11]. The dynamic complexity is a measure of complexity of the subset of the code that is actually executed when the system is performing a given function.

While a program is executing any one of its many functionalities, it apportions its time across one to many program components, depending on the execution profile. The execution profile for a given functionality is the proportion of time spent in each program component during the time that function was expressed [6]. The p_k can be calculated from simulation results obtained by running the CPN model of the system. All the dynamic measures of system complexity are based on the results obtained from the dynamic simulation of the CPN model (simulation report) of the system. In such a report, the information can be extracted regarding the dynamic behavior of the system, e.g., number of times each function is fired (invoked). This information is used to calculate p_k.

a) Functional Complexity (fcpx, FCPX): Once p_k is calculated, the fcpx of component k in the system running an application for an execution profile is:

$$\text{fcpx}_k = \text{sscpx}_k \cdot p_k. \tag{8}$$

Similar to static complexity,

$$\text{FCPX} = \sum_k \text{fcpx}_k. \tag{9}$$

b) Operational Complexity (ocpx, OPCX): In the course of execution, changes in the functionalities in a running program cause the program to select only a subset of possible paths from the set of all possible paths through the control flow graph.

As presented in [6], when a distinct functionality of a program is exercised, a subset or subgraph of the program executes. This subgraph has its own complexity representing the complexity of just the code that was executed. This metric clearly cannot be greater than the static complexity of the program.

Apply this measure to nonprimitive components (components modeled by a CPN sub-model) in the specification model using scenario-based simulations. During the simulation, only a part of the flow graph of the CPN sub-model representing the nonprimitive component is activated. This flow graph is determined by the sequence of transitions fired. For each simulation scenario, measure the dynamic cyclomatic complexity (VG') for this part of the flow graph. From the simulation reports, it is possible to observe what transitions have been fired and hence the dynamic cyclomatic complexity can be calculated.

Once the dynamic cyclomatic complexity is computed, then:

$$c_k = \frac{\text{VG}'_k + 1}{\text{VG}_k + 1}. \tag{10}$$

VG in Section IV-A1, used in measuring static complexity, represents the cyclomatic complexity of a single primitive transition. The flow graph used to calculate VG_k is based on the CPN sub-model of the nonprimitive transition. The c_k represents the fraction of the flow graph that was activated during the simulation; for a primitive component $c_k = 1$.

$$\text{ocpx}_k = p_k \cdot c_k \cdot \text{SCPX}_k. \tag{11}$$

SCPX_k is the relative static complexity of nonprimitive component K in the system.

The OCPX is the sum of the operational complexities of the components it contains:

$$\text{OCPX} = \sum_k \text{ocpx}_k. \tag{12}$$

c) Concurrence Complexity (ccpx, CCPX): Today's systems frequently use concurrent processes in a real time environment. Functional complexity and operational complexity (described in the previous sections) do not account for concurrence in a system, which is another aspect of the system's dynamic behavior. In other words, they do not capture the effects of concurrence with its clearly added complexity factors in the system development & operation. However, when dealing with concurrence, the state space of the system can be very large. This section presents a method to measure the complexity added to the

system due to concurrence in the software system. This measure is 'concurrence complexity', a new complexity metric for software systems.

Simulation reports are used to measure the complexity due to the concurrence added to the system. Concurrence does not exist for single components (represented by nonprimitive transitions); therefore concurrence complexity is calculated for each nonprimitive component. From the simulation reports, measure the degree of concurrence using CCF_k; it is defined as the maximum number of concurrently active transitions of component k at any given time during the simulation.

The rationale for choosing the maximum number of active transitions to determine the complexity is discussed here. Assume that: a) the number of active transitions at any given time determines the state of a concurrent component; and b) the size of the state space depends on the maximum number of active transitions. The size of the state space grows exponentially with this maximum number of active transitions. Because the complexity of the dynamic behavior depends not only on the number of active components at any given time, but also on the dependencies between these components, the maximum number of active components should be used to determine the complexity. This number is calculated for each nonprimitive component consisting only of primitive components, which are typically at the lower levels of the hierarchy in large complex systems.

Define:

$$\mathrm{ccpx}_k \equiv \frac{\mathrm{CCF}_k - 1}{F_k} \cdot \mathrm{fcpx}_k. \tag{13}$$

$F_k \equiv$ number of distinct fired transitions in component k throughout the whole simulation time.

If there is only one process active (or running) in the system, there is no concurrence. Hence subtract 1 from the CCF to get the number of sub-components executing at the same time.

$$\mathrm{CCPX} = \sum_k \mathrm{ccpx}_k. \tag{14}$$

B. Severity

Severity is a procedure by which each potential failure mode is ranked according to the consequences of that failure mode. According to MIL-STD-1629A, severity considers the worst potential consequence of a failure, determined by degree of injuries or system damages that ultimately occur.

FMEA is a systematic approach which, on a component-by-component basis, details all possible failure modes, and identifies their effects on the system [12]. It is used to perform single-random-failure analysis as required by IEEE STD 279-1971, 10 CFR 50. FMEA is applied after the initial system designs are completed.

System-safety requirements can be analyzed by using FMEA. PN inherently have all the features required of an FMEA analysis tool since they can be used to determine the effects of a failure in a component. By injecting failures in a component during the simulation of the model, one can precisely study the failure propagation throughout the system.

Ranking severity is rated in 'more than one way' and 'more than one purpose' [13]. This study used the severity classifications recommended by MIL-STD-1629A:

Catastrophic: A failure that can cause death or system loss

Critical: A failure that can cause severe injury, or major system-damage that results in mission loss.

Marginal: A failure that can cause minor injury, or minor system-damage that results in delay or 'loss of availability' or 'mission degradation'.

Minor: A failure not serious enough to cause injury, or system damage, but that results in unscheduled maintenance or repair.

Based on the effects observed after injecting faults to mimic the failure of system components through dynamic simulation of the model, we assign severity indices of 0.25, 0.50, 0.75, 0.95 to minor, marginal, critical, catastrophic classes respectively. A severity index is used for hrf calculations.

C. Heuristic Risk Factor (hrf, HRF)

The objective of risk assessment is to classify the system-function requirements according to their relative importance in terms of such factors as severity and complexity. We define hrf as the measure of risk. There is a strong relation between software quality and the complexity of the components comprising it.

On the other hand, we must assess the effects of failures in the critical components of the system. Severity of a component-failure affects the quality of the software too. Thus these 2 aspects are to be considered for overall risk assessment in a (software) system. Hence, we define the hrf which takes considers the complexity & severity of components in a system as:

$$\mathrm{Hrf}_k = \mathrm{cpx}_k \cdot \mathrm{svrty}_k, \quad 0 \le \mathrm{cpx}_k \le 1, \quad 0 \le \mathrm{svrty}_k \le 1 \tag{15}$$

The normalized complexity metric of component k is obtained by dividing the component complexity by the complexity value of the highest complexity component in the system specification or across several system specifications if more than one system specification model is being analyzed.

$$\mathrm{HRF} = 1 - \prod_k (1 - \mathrm{hrf}_k). \tag{16}$$

V. Risk Assessment of the EOS Commanding Specifications

This section presents the risk-assessment results for the specification models of Commanding. These results are obtained by applying the methods in Section IV. As mentioned in Section IV, the only difference between the sequential model and the pipeline model is in the specification of the EOC-CONT. The method in this section can monitor this apparently minor difference as shown in this section.

A. Complexity Analysis

Static complexity is presented first because it is the starting point of complexity analysis. This is followed by dynamic com-

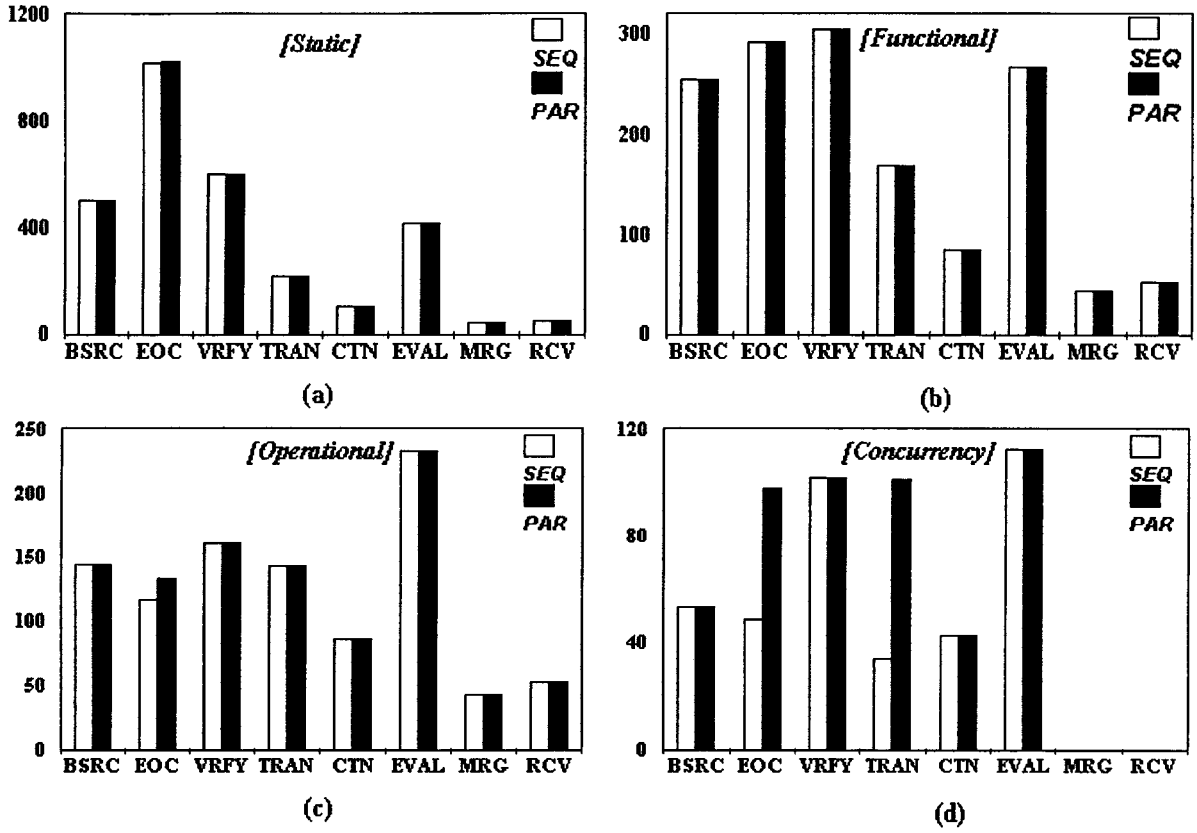


Fig. 5. Complexity: (a) Static, (b) Functional, (c) Operational, (d) Concurrence.

plexity analysis based on functional and operational complexities. Concurrence complexity analysis is presented last.

1) Static Complexity: The sample data-set on the complexity measures of primitive transitions (based on M, N, and VG_t) in both the sequential and the pipeline models are filtered to eliminate redundancy. The standardized matrix of the sample data-set is computed as described in [1]. Applying principal component analysis [33], the eigenvectors matrix is produced and is used to calculate the static complexity of each primitive transition. The static complexity of a nonprimitive function SCPX is obtained by summing $(\mathrm{scpx}_k, k = 1, \ldots, n)$ the static complexities of its n primitive transitions. Fig.5(a) shows the calculated static complexities at the function level for the components of the sequential and pipeline models.

As anticipated, the static complexity measures of the two models are very close since there is no important difference between the two models statically. However, as the dynamic behavior of the two systems is used, we expect discovering some difference as shown in Sections V-A2, V-A3, and V-A4.

2) Functional Complexity: The dynamic measures of system complexity are based on the results obtained from the dynamic simulation of the CPN model. The simulation reports from both the sequential and the pipeline model running the same scenario are analyzed. These reports are used to calculate the probability of each transition being executed during the simulation. Once these probabilities are determined, the fcpx_k of specification component k is computed using (8).

Fig5(b) shows the functional complexities at the function level for sequential and pipeline models.

Because the operational profile for both models is the same, we anticipate the same functions in both the systems to participate in the execution: the same functions are exercised in both systems. As Fig.5(b) shows, functional complexity does not distinguish between the two systems, although some minor difference is observed for EOC-CONT.

3) Operational Complexity: The functional complexity metric does not reveal the difference between the sequential model and the pipeline model. Since we are running the same execution profile in the same environment, the 'components being executed' and 'their probabilities of execution' remain the same for both models. The operational complexity metric is based on the cyclomatic complexity VG' of the subgraph being executed during the simulation is shown in (10) & (11).

Fig.5(c) shows the operational complexities at the function level for sequential & pipeline models; all the components demonstrate the same operational complexity and again the only slight difference between the two systems is in EOC-CONT. Nevertheless, operational complexity seems to be a better metric compared to static & functional complexity measures. The slight difference between the operational complexity of the sequential model and the pipeline model is because there is a difference between the static complexity of the two models (see Fig.5(c)). Because the difference in the static complexity of the two models is small (coming from the difference in the VG of the EOC-CONT), the operational complexity does not show any important difference.

4) Concurrence Complexity: Complexity measures obtained so far can not detect the difference between the sequential

& pipeline models. However, from the simulation reports, in the pipeline model several more processes are running at the same time compared to the sequential model. This concurrence increases the dynamic complexity of the system. In other words, the dynamic behavior of the system is such that the system is building a command, verifying another command, merging some other command, and transmitting a command—all at the same time.

Fig.5(d) demonstrates the difference between the sequential model and the pipeline model of the system. The concurrence complexity, introduced in this paper, is a major aspect of complexity measure in analyzing the models. Despite the fact that all the components and code segments are the same, this metric has been able to detect the difference between the two models from the dynamic behavior of the system.

B. Severity Analysis

To accomplish the risk-assessment task, we still need to analyze the severity associated with the system components. Basic failure modes of each system component and their relevant effects on the overall system operation are studied & considered. Numbers are then assigned to each class of severity based on the 'degree of severity' and 'criticality of their failures on the system' as described in Section IV-B.

The CPN model of the system specification is used to study the effects of the failures on a component-by-component basis in the two systems. Using the CPN model as a tool for FMEA, faults (one at a time) can be injected into the components of the systems. Then the simulator is used to study the effects of failures. Since the exact reaction of the spacecraft is unknown (e.g., in response to receiving out-of-sequence commands or corrupted commands), assumptions are made whenever required. These assumptions, however, could be agreed upon by the domain expert(s). For example it is assumed that if the spacecraft receives correct commands that are out of sequence, then it could get into a critical situation.

The following few cases show examples of severity analysis for some of the components or subcomponents of the specification models described in Section III. The cases start with the component name in the model, their determined severity level, the specification models considered, and a brief explanation of the analysis.

AOI & VC are subcomponents of BSRC;
VC1 & VC2 are subcomponents of VC.

1)

AOI MINOR
(SEQUENTIAL & PIPELINE)

AOI failed to produce GOOD command ⇒ BAD command is generated ⇒ BAD command will be rejected by VC ⇒ no hazardous situation is anticipated ⇒ unanticipated-maintenance.

2)

VC1 & VC2 CATASTROPHIC
(SEQUENTIAL & PIPELINE)

VC1 or VC2 fails to rectify a BAD command ⇒ spacecraft will receive the BAD command ⇒ BAD command could lead to loss of the spacecraft.

3)

CTN MAJOR
(SEQUENTIAL & PIPELINE)

Early TLR (Transmission Limit Reached) ⇒ next command is uplinked ⇒ previous command execution/uplink status is evaluated after the newly uplinked command is executed ⇒ status of the previous command is evaluated for the current command ⇒ Major damages to the system are probable.

4)

MRG CRITICAL
(SEQUENTIAL & PIPELINE)

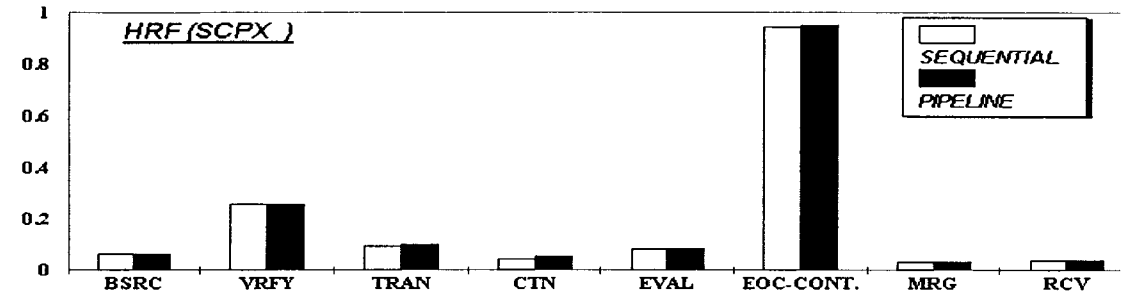
Fails to merge VRC and VPC ⇒ corrupted UDS is generated and uplinked to the spacecraft ⇒ Critical situation could happen.

C. Risk Assessment

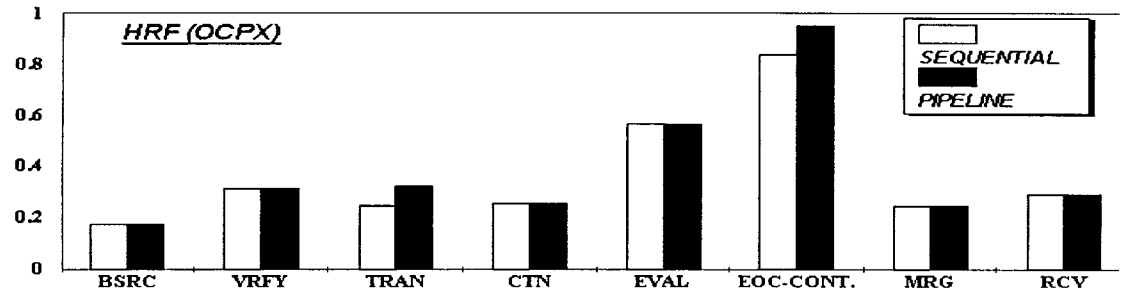
Based on the values calculated for complexity & severity of the system components, compute the hrf as the final stage in the risk assessment procedure. Before applying our methodology, normalize the complexity measures: static, functional, operational, and concurrence.

Four hrf are calculated for this example. The hrf #1 reflects the effect of the static complexity; hrf #2 accounts for the functional complexity measures; hrf #3 is the result of operational complexity. These three measures of risk factor are based on the component level. Risk factors, hrf #4, associated with the concurrence complexity are calculated at the function level since there is no concurrence complexity at the component level, and we calculated concurrence complexity at the function level. However, the severity index at the function level is calculated based on the severity of the components comprising the function.

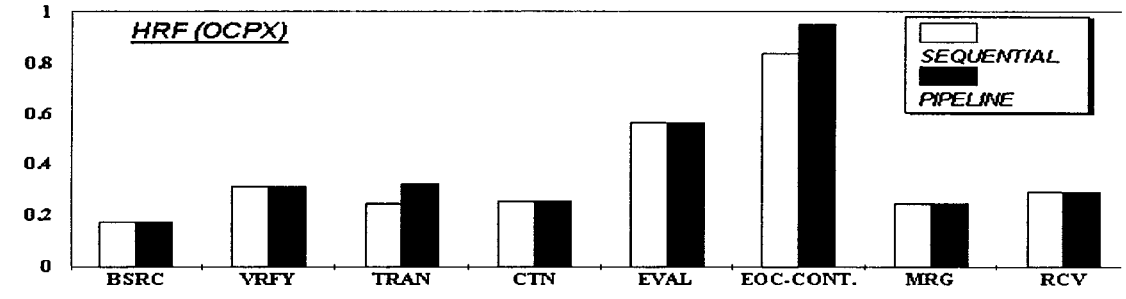
The graphs in Fig.6 clearly demonstrate the effectiveness of our methodology in assessing the risk of the sequential & pipeline systems. As mentioned in the previous paragraph, 4 hrf are calculated, each relating to one complexity measure. The hrf based on static complexity, in Fig.6(a), does not distinguish between the sequential & pipeline models. All the subsystems present very close hrf. Fig.6(b) shows the hrf based on the functional complexity, which detects a difference in TRAN. Nevertheless, the difference in EOC-CONT in the two models is still not important. Calculating hrf based on the operational complexity gives better results. This difference is coming from the operational complexity. In Fig.6(d), the difference between the two models is noticeable. The hrf of TRAN in the pipeline model is more than twice that of the sequential model. EOC-CONT in the pipeline model shows a higher risk factor. This stems from the fact that in the pipeline model, EOC-CONT, that orchestrates the system operations, has



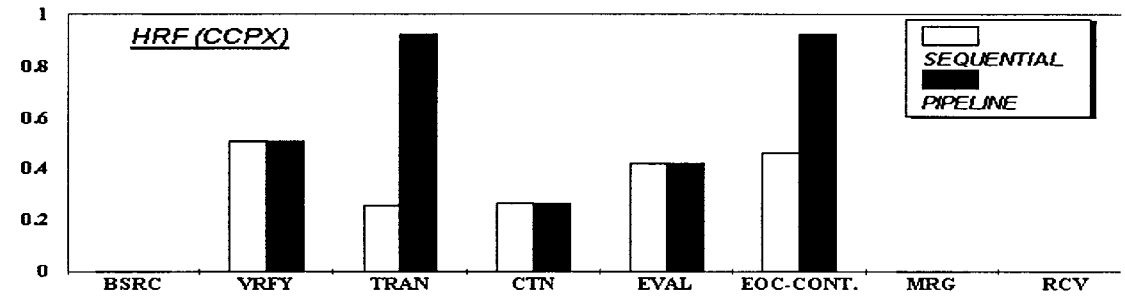
a) HRF based on static-complexity (scpx) measure



b) HRF based on functional-complexity (fcpx) measure



c) HRF based on operational-complexity (ocpx) measure



d) HRF based on concurrence-complexity (ccpx) measure

Fig. 6. Risk Assessment Graphs.

greater complexity due to the greater parallelism. Obviously failures of this component can jeopardize the spacecraft.

As shown in Section V-A, methods based on the conventional complexity metrics cannot differentiate between the two models even though functional complexity and operational complexity are based on the dynamic behavior of the system. The hrf based on concurrence complexity measures captures the differences in the dynamic behavior of the system specifications. Using results obtained by applying the methodology in Section IV, analysts can clearly decide which components and/or subsystems require more development resources.

D. How the Results Can be Used

Components with an hrf, of any type larger, than a certain threshold level such as 0.4 can be designated as high-risk components. In our case study, these components include EOC-CONT, EVAL, and VRFY in both specifications as well as the TRAN component for the pipelined specification. The EOC-CONT has clearly high-risk, based on all 4 risk factors. The EVAL component exhibits a relatively high hrf based on operational complexity, whereas VRFY exhibits a relatively high hrf based on concurrence complexity.

One might conclude that the TRAN component is not a high-risk component, using the graphs on hrf based on conventional complexity analysis in Figs.6(a)–(c). However, the hrf analysis based on the new concurrence complexity measure in this paper, and in Fig.6(d), leads to a different conclusion. It is clear that TRAN in the pipeline specification is a high-risk component since it has a high hrf based on concurrence complexity.

This discussion shows that analysts should not rely on a risk-factor based on one complexity measure. The risk factors based on the complexity measures calculated using the methods in this paper, provide multiple prospectives for risk assessment in which a high-risk component can be identified if any of its risk factors is higher than a given threshold.

The identification of high-risk components is particularly important not only for the development process but for independent verification & validation (IV&V) processes. In this case the IV&V resources are focused on these special components to identify possible problems which can be tackled by developers at the early stages of development.

REFERENCES

[1] H. H. Ammar, T. Nikzadeh, and J. B. Dugan, "A methodology for risk assessment of functional specifications using colored Petri nets," presented at the Proc. Fourth Int'l Software Metrics Symp. (Metrics'1997), Nov. 1997.

[2] J. C. Munson and T. M. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Trans. Software Eng.*, vol. 18, no. 5, pp. 423–433, 1992.

[3] R. W. Selby and V. R. Basili, "Analyzing error-prone system structure," *IEEE Trans. Software Eng.*, vol. 17, no. 2, pp. 141–152, 1991.

[4] M. Z. Wayne and D. M. Zage, "Evaluating design metrics on large scale software," *IEEE Software*, vol. 10, no. 7, pp. 75–81, 1993.

[5] B. A. Kitchenham and L. Pickard, "Toward a constructive quality model," *Software Eng. J.*, vol. 2, no. 7, pp. 114–126, July 1987.

[6] *Handbook of Software Reliability Engineering*, McGraw-Hill, 1995. "Software metrics for reliability assessment".

[7] ——, "The dimensionality of program complexity," in *Proc. 11th Annual Conf. on Software Eng.*, Pittsburgh, PA, May 1989, pp. 245–253.

[8] T. J. McCabe, "Complexity metrics," *IEEE Trans. Software Eng.*, vol. 2, no. 4, pp. 308–320, Dec. 1976.

[9] G. A. F. Seber, *Multivariate Observations*: John Wiley & Sons, 1984.

[10] J. C. Munson and T. M. Khoshgoftaar, "Applications of a relative complexity metric for software project management," *J. Systems and Software*, vol. 12, no. 3, pp. 283–291.

[11] T. M. Khoshgoftaar, J. C. Munson, and D. L. Lanning, "Dynamic system complexity," in *Proc. Int'l Software Metrics Symp.*, Baltimore, MD, May 1993, pp. 129–140.

[12] *Procedures of Performing a Failure Mode, Effects and Criticality Analysis*. US DoD.

[13] H. Kumamoto and E. J. Henley, *Probabilistic Risk Assessment for Engineers and Scientists*, 2nd ed: IEEE Press., 1996.

[14] F. Belli and J. Dreyer, "Systems specification, analysis, and validation by means of timed predicated/transition nets and logic programming," in *Proc. Int'l Symp. Software Reliability Eng.*, Oct. 1995, pp. 68–77.

[15] M. H. Halstead, *Elements of Software Science*, Elsevier, North-Holland, 1977.

[16] S. H. Kan, *Metrics and Models in Software Quality Engineering*: Addison Wesley, 1995.

[17] C. Ebert, "Evaluation and application of complexity-based criticality models," in *Proc. 3rd Int'l. Software Metrics Symp.*, Berlin, Mar. 1996, pp. 174–184.

[18] T. M. Khoshgoftaar and J. C. Munson, "Predicting software development errors using software complexity metrics," in *Software Reliability and Testing*: IEEE Computer Society Press, 1995, pp. 20–28.

[19] D. I. Heimann, "Using complexity-tracking in software development," in *Proc. Annual Reliability and Maintainability Symp.*, 1995, pp. 433–437.

[20] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel, "Early quality prediction a case study in telecommunications," *IEEE Software*, pp. 65–71, 1996.

[21] N. Leveson and J. L. Stolzy, "Safety analysis using Petri nets," *IEEE Trans. Software Eng.*, vol. SE-13, no. 3, pp. 386–397, Mar. 1987.

[22] J. Breger and L. Lamontagne, "A colored Petri net model for a naval command and control system," presented at the Proc. 14th Int'l Petri Net Conf., vol. 691, Chicago, IL, 1993.

[23] C. Capellmann and H. Dibold, "Petri net based specifications of services in an intelligent network," in *Proc. 14th Int'l Petri Net Conf.*, vol. 691, Chicago, IL, 1993, pp. 542–551.

[24] L. Cherkasova, V. Kotov, and T. Rokicki, "On net modeling of industrial size concurrent systems," in *Proc. 14th Int'l Petri Net Conf.*, vol. 691, Chicago, IL, 1993, pp. 552–561.

[25] ——, "On scalable net modeling of OLTP," in *Petri Nets and Performance Models*, PNPM93, 1993, pp. 270–279. Proc. 5th Int'l Workshop.

[26] S. Christensen and L. O. Jepsen, "Modeling and simulation of a network management system using hierarchical colored Petri nets," in *Modeling and Simulation Proc*, Copenhagen, 1991, pp. 47–52. European Simulation Multiconf.

[27] H. Clausen and P. R. Jensen, "Validation and performance analysis of network algorithms by colored Petri nets," in *Proc. 5th Int'l Workshop*, Toulouse, 1993, pp. 280–289.

[28] D. J. Floreani and J. Billington, "Designing and verifying a communications gateway using colored Petri nets and design/CPN," in *Proc. 17th Int'l Petri Net Conf.*, vol. 1091, Osaka, 1996, pp. 153–171.

[29] H. J. Genrich and R. M. Shapiro, "Formal verification of an arbiter cascade," in *Proc. 13th Int'l Petri Net Conf.*, vol. 616, 1992, pp. 205–224.

[30] P. Huber and V. O. Pinci, "A formal executable specification of the ISDN basic rate interface," in *Proc. 12th Int'l Conf. Application and Theory of Petri Nets*, 1991, pp. 1–21. Aarhus.

[31] J. B. Jergensen and K. H. Mortensen, "Modeling and analysis of distributed program execution in BETA using colored Petri nets," in *Proc. 17th Int'l Petri Net Conf.*, vol. 1091, Osaka, 1996, pp. 249–268.

[32] W. W. McLendon and R. F. Vidale, "Analysis of an Ada system using colored Petri nets and occurrence graphs," in *Proc. 13th Int'l Petri Net Conf.*, vol. 616, Sheffield, 1992, pp. 384–388.

[33] T. Nikzadeh, "Risk Assessment and Complexity Analysis of Software Systems Using Colored Petri Nets," Master's thesis, Electrical and Computer Eng. Dept, West Virginia Univ., Aug. 1997.

[34] *Functional and Performance Requirements Specification for the Earth Observing System Data and Information System (EOSDIS) Core System*, Nov. 1994. Hughes Applied Information Systems, rev. A and CH-01.

[35] *Flight Operation Segment (FOS) Requirements Specification for the ECS Project*, Nov. 1994. Hughes Applied Information Systems, General Requirements.

[36] B. Mikolajczak and J. Rumbut, "A systematic method of object-oriented software design using colored Petri nets," in *Naval Underwater Warfare Ctr, 14th Int'l Conf. Applications and Theory of Petri Nets*, Chicago, IL, June 1993, pp. 21–25.

[37] U. Buy and R. H. Sloan, "Analysis of real-time programs with simple time Petri nets," in *Proc. 1994 Int'l Symp. Software Testing and Analysis*, Aug. 1994, pp. 228–239.

[38] K. Lateef, H. H. Ammar, V. Mogulotho, and T. Nikzadeh, "A methodology for verification and analysis of parallel and distributed systems requirements specifications," presented at the 2nd Int'l Workshop Software Eng. for Parallel and Distributed Systems, Boston, MA, May 1997.

[39] H. Ammar, T. Nikzadeh, and J. B. Dugan, "Risk assessment of functional specification of software systems using colored Petri nets," presented at the Proc. Int'l Symp. Software Reliability Eng. (ISSRE'97), Nov. 1997.

[40] Design/CPN [Online]. Available: http://www.daimi.au.dk/designCPN/

Hany H. Ammar received his B.S.E.E. (1975) in electrical engineering and a B.S. (1977) in physics from Cairo University, Egypt; his M.S. (1980) in electrical engineering from the University of Texas at El Paso; and his Ph.D. (1985) in electrical engineering from the University of Notre Dame. He is a Professor of Computer Engineering in the Department of Computer Science and Electrical Engineering at West Virginia University, and was an Assistant Professor of Computer Engineering at Clarkson University. He has performed and directed research in software engineering in the areas of software specification and analysis using Petri nets; object-oriented analysis and design of real-time systems; software metrics; risk assessment; performance, reliability, and performability analysis of software specifications and designs; component-based software architecture; design patterns and frameworks; and high performance computing with applications to neural networks and image processing. Dr. Ammar has served in the organization committees and program committees of several International Conferences and Workshops. He is a member of the IEEE and ACM, Eta Kappa Nu, and Tau Beta Pi.

Tooraj Nikzadeh received his B.S.E.E. (1985) in electrical engineering from Sharif University of Technology (Iran), and M.S. (1997) in computer engineering from West Virginia University at Morgantown. He is working as a senior programmer/manager at 24/7 Meida/IMAKE Software and Services, Inc. As research assistant he has done research on software engineering in risk assessment, performance, reliability, and performability analysis of software specifications and designs under the supervision of Dr. Hany H. Ammar. He has researched in deploying soft computing (deploying neural networks, fuzzy-expert systems) in building decision support systems, and system simulations in petroleum and mining engineering. He has several years of working experience in the IT industry designing and developing Internet-based distributed software systems. He has several publications in his related field of research.

Joanne Bechta Dugan was awarded the B.A. in mathematics and computer science from LaSalle University, Philadelphia (1980), and the M.S. and Ph.D. in electrical engineering from Duke University, Durham (1982 and 1984). Dr. Dugan is Professor of Electrical and Computer Engineering at the University of Virginia. She has performed and directed research on the development and application of techniques for analysis of computer systems which are designed to tolerate hardware and software faults. Her research interests thus include hardware and software reliability engineering, fault tolerant computing, and mathematical modeling using dynamic fault trees, Markov models, Petri nets and simulation. Dr. Dugan is an IEEE Fellow, was Associate Editor of this *IEEE Trans. Reliability* for 10 years, and is Associate Editor of the *IEEE Trans. Software Engineering*. She served on the National Research Council Committee on Application of Digital Instrumentation and Control Systems to Nuclear Power Plant Operations and Safety.