基于系统多维要素的安全关键软件验证方法

吕小虎 韩笑冬 宫江雷 王志杰 刘小鲲
(中国空间技术研究院通信卫星事业部 北京 100094)

摘 要 软件密集型系统已成为发展的必然趋势。安全关键软件功能的比重持续上升,与之相关的安全性问题也日益凸显,且问题的影响因素呈现复杂、多维、动态、隐蔽等特征。因此,寻求合理的验证方法成为了迫切需要,而如何对其进行有效验证,也成为软件安全性工作的难点。结合安全关键软件的研制工作,文中研究并提出基于系统多维要素的安全关键软件验证方法,从系统的角度建模对影响软件安全性的多维危险要素;在此基础上,通过构建安全关键软件的需求约束集和验证集,给出具体的验证方法和步骤。实际应用表明,与传统的局限于软件逻辑自身的验证方法相比,文中所提方法能够有效识别大量软件潜在的深层次的问题。

关键词 安全关键软件,多维要素,约束集,验证集
中图法分类号 TP311.52 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.09.022

Systemic Muti-factors Based Verification Method for Safety-critical Software

LV Xiao-hu HAN Xiao-dong GONG Jiang-lei WANG Zhi-jie LIU Xiao-kun
(Institute of Telecommunication Satellite, China Academy of Space Technology, Beijing 100094, China)

Abstract Software-intensive systems have been the inexorable development trend. The proportion of functions of safety-critical software keep growing, and the software safety problems are highlighted increasingly, in which the influence factors are characterized by complex, multidimensional, dynamic and insidious. Therefore, it's urgent to seek a reasonable verification method for safety-critical software, and how to effectively verify it has become a difficult issue in software safety-related work. Based on the research and development of safety-critical software, this paper studied and proposed a verification method for safety-critical software based on systemic muti-factors, modeled the muti-factors that affect software safety from the point of system, and gave detailed verification methods and steps through constructing the requirement constraint sets and verification sets. The results of practical application show that the proposed method can effectively identify potential and systemic problems in safety-critical software compared with the traditional verification methods limited to software logic.

Keywords Safety-critical software, Muti-factors, Constraint sets, Verification sets

1 引言

2005年,TNO/IDATE报告指出,自2002－2015年间,世界军工产业中软件研制比重将由35%提升至45%,软件密集型系统成为必然趋势,如F22飞机上嵌入式系统代码多达300万行,F35机载和地面的嵌入式系统代码多达1 500万行[1]。

随着软件执行功能比例的持续上升,软件代码规模不断增长,自身结构复杂度的提高且软硬件耦合程度的增强,使得与软件相关的安全性问题日益凸显。如德国汉莎航空公司的飞机在滑行过程中,由于未收到前轮着地信号,不满足相关制动条件,导致飞机制动控制软件没有进行制动[2];此外,某卫星由于空间单粒子效应引起有效载荷管理单元未能正确转发GPS定位广播数据,从而导致天线转动异常。

GJB/Z 102A-2012[3]指出,软件安全性是指软件运行不引起系统事故的能力,安全关键软件是指其错误可能导致系统发生严重危险的软件。

大量安全事故表明,明确安全关键软件安全性问题的薄弱环节和问题机理,聚焦软件研制过程中对于安全性的深层次需求,寻求合理的验证方法成为了迫切需要,而如何对软件进行分析和验证也成为软件安全性工作的难点。传统的软件安全性验证方法采用从类似于硬件失效问题的视角看待软件问题,认为导致软件相关事故的原因中,软件自身的不可靠因素占据很大比重,忽略了许多软件相关的事故其原因是多维的、系统的这个关键问题,导致难以挖掘潜在的、深层次的软件安全性问题。

到稿日期:2018-07-30 返修日期:2018-10-27 本文受国家自然科学基金(61471360)资助。
吕小虎(1983－),硕士,高级工程师,主要研究方向为嵌入式软件设计、安全关键软件设计与验证,E-mail:xibeixiongying11@163.com(通信作者);**韩笑冬**(1983－),博士,高级工程师,主要研究方向为嵌入式软件设计、智能控制;**宫江雷**(1986－),硕士,高级工程师,主要研究方向为嵌入式软件系统级设计;**王志杰**(1970－),硕士,高级工程师,主要研究方向为测控软件系统级设计;**刘小鲲**(1982－),硕士,工程师,主要研究方向为嵌入式软件测试。

结合大量安全关键软件研制工程实践，文中研究并提出基于系统多维要素的安全关键软件验证方法。与传统的仅局限于软件自身逻辑进行安全性验证的方法相比，所提方法能够从系统的角度对影响软件安全性的多维危险要素进行解析和建模，并能够形成安全关键软件验证的方法和处理步骤，还可以结合具体实例验证方法的可行性。

2　相关研究工作

围绕安全关键软件的安全性，国内外已开展了相关研究，提出了一些典型的标准和相关理论。

GJB 900/Z 142-2004[4]针对安全关键软件的安全性分析流程与方法给出了详尽描述。此外，文献[5]对软件安全性内涵与外延进行了深入剖析，给出了软件安全性的度量模型，并基于软件工程视角对软件安全性开发、设计、评估等方面进行了综述；文献[6]基于嵌入式机载软件，介绍了相关领域的标准，给出了软件安全性保证的三方面工作，即安全需求的提约、安全关键软件的研制以及对软件安全需求的验证。

NASA 于 2013 年更新了发布的 NASA-STD-8719.13《软件安全性标准》[7]，配套的指南为 NASA-GB-8719.13C《NASA 软件安全性指南》。在该标准中，对安全关键软件的属性特征进行了详尽说明，并对软件开发各阶段的安全性分析验证要求以及方法进行了简要说明。

2005 年，美国麻省理工大学 Leveson 教授提出了基于系统理论的危险建模与处理模型(Systems-Theoretic Accident Modeling and Processes，STAMP)，首次从系统的角度分析了系统各部分之间及与外界的关联关系，并将人、环境等多种安全约束因素纳入综合考虑。基于 STAMP 模型，Leveson 等于 2007 年进一步提出了一种新的危险分析技术 STPA(基于 STAMP 的危险分析)[8]并研制了系统级需求建模工具 Spec-TRM[9]，给出了针对安全性关键系统危险演化机理系统分析的流程框架，并对安全关键软件参与的控制过程进行了状态建模；该理论和技术解决了长期以来脱离软件层面解决软件安全性问题的弊端，已在美国航天飞机的控制系统、日本的航天器研制[10]中得到了成功应用。

3　基于系统多维要素的安全关键软件验证方法

3.1　软件安全性问题机理分析及多维影响要素模型的构建

软件密集型系统中，围绕安全关键软件产生的问题的复杂性体现在：除软件自身的设计可能存在缺陷外，更多的是系统环境中包含了系统需求不完善、系统各模块之间交互异常、外部环境干扰、人员操控失误[11-13]等诸多因素，导致在系统运行过程中，背离了安全性约束，从而产生危险因素，将这些因素通过软件进行传递，最终产生系统性问题，甚至引发安全事故。

与软件相关的安全性约束因素呈现多维、动态、隐蔽的特征，系统性地剖析问题机理，提炼影响安全关键安全性的要素，并在此基础上采取有效的手段进行识别验证，是解决软件安全性问题的首要前提。

通过对安全关键软件研制工作的梳理，本文从系统的角度出发，围绕软件自身逻辑及其运行环境、交互特性等方面，将系统中影响安全关键软件的要素抽象为涵盖以下 5 个元素的结构化模型：

$$F=\langle PE,HW,EN,SW,OT\rangle$$

其中，F 表示软件安全性影响要素结构化模型；PE 表示人机交互要素；HW 表示软硬交互要素；EN 表示运行环境要素；SW 表示软件自身算法及实现要素；OT 表示其他约束要素。

在此基础上，对多维安全影响要素进行系统建模，如图 1 所示。

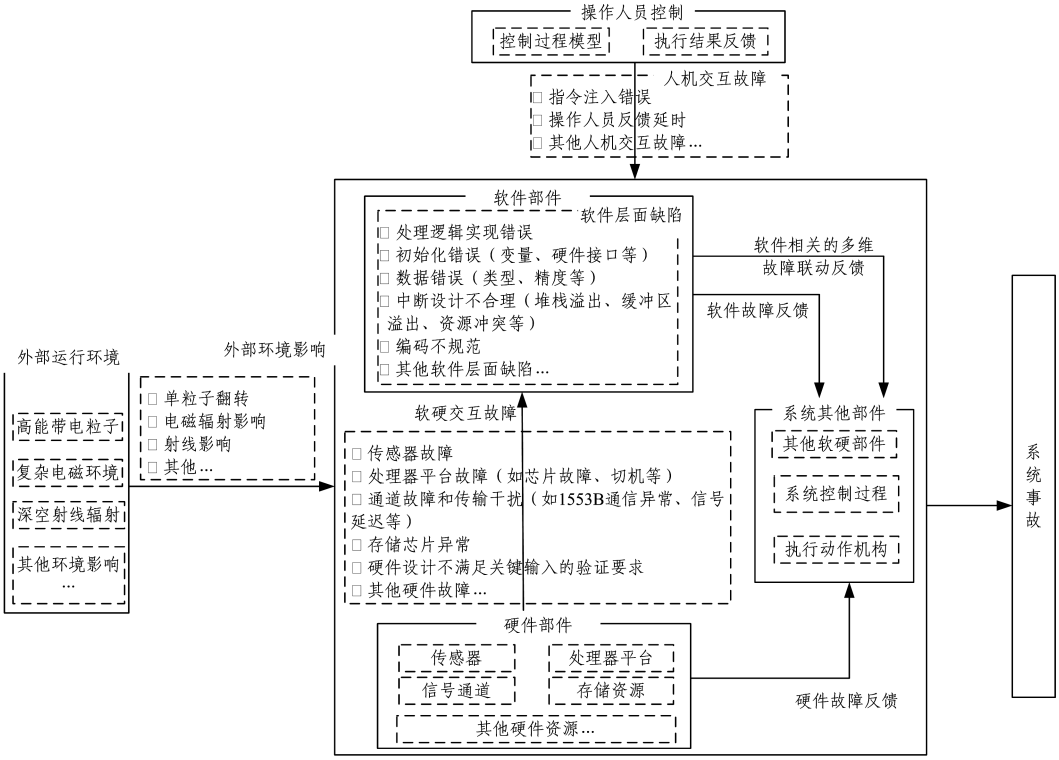

图 1　软件密集型系统中的多维因素模型

Fig. 1　Muti-factors model in software-intensive systems

上述 5 个要素与安全关键软件问题的内在关联关系的解析如下。

PE(人机交互要素)主要体现在操作人员指误控制发令、软件系数设置不合理、操作人员对计算结果反馈延时等;

EN(运行环境要素)主要包括空间强电磁干扰、深空射线辐射、高能粒子干扰等;

HW(软硬交互要素)主要包括:硬件故障(如切机、掉电等),控制循环中的信号传输延迟,通信总线故障,软硬件交互时序异常,执行动作机构未在规定的时间内响应外部指令或处理关键参数等;

SW(软件自身算法及实现要素)主要包括:处理逻辑实现不符合需求规定,初始化错误,数据处理错误(数据有效性判断、数组越界、精度等),中断设计不合理(中断初始化及清零时机、堆栈溢出、资源冲突等),逻辑分支处理缺陷(存在未处理逻辑分支)等;

OT(其他约束要素)主要是指软件运行模式切换、运行状态更改过程中需要考虑的约束要素。

基于上述模型,当软件运行在某种特定运行状态 OT_i 时,存在某种特定的软硬交互 HW_i 及人机交互 PE_i,并且受某种运行环境 EN_i 的影响;同时软件自身实现逻辑问题 SW_i 等诸多要素的共同作用(这些要素中某些要素可能为空),可能导致某种特定的危险事件的发生,则 $F_i=\langle PE_i, HW_i, EN_i, SW_i, OT_i\rangle$ 成为影响安全关键软件的一个要素结构。

3.2 基于多维要素的安全关键软件验证方法

在完成对影响安全关键软件安全性影响要素模型的构建的基础上,结合安全关键软件的实际运行处理过程,本文给出基于多维要素的安全关键软件验证方法,该方法分为安全关键软件需求约束集的构建和安全关键软件验证集的设计两个步骤。

3.2.1 安全关键软件需求约束集的构建

国外研究指出,70%~90%的与软件安全性相关的决策都需要在系统需求阶段做出[14-15],若未将危险分析整合到系统需求设计中,将会引入很多未知风险。因此,必须在系统的需求阶段纳入对安全关键软件的安全性需求考虑,并建立安全关键软件需求约束集。构建软件安全性需求集可分解成以下步骤进行。

Step1 对任务的安全性处理模型进行抽象。

结合 3.1 节建立的软件安全性影响要素模型,借鉴 Leveson 教授提出的 STAMP 模型处理控制反馈结构,对安全关键软件参与的系统控制处理过程进行抽象,形成闭环的任务安全性处理模型,如图 2 所示。在该模型中,运行于特定硬件和运行环境中的软件模块以操作人员的控制指令、外部采集器提供的数据以及动作执行机构提供的控制变量反馈为处理输入,通过算法处理形成处理输出和显示信息,分别反馈至动作执行机构和操作人员。

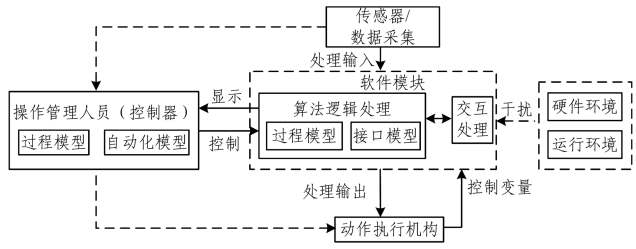

图 2 基于安全关键软件的控制过程处理模型

Fig. 2 Control process model based on safety-critical software

Step2 建立处理模型中所有的安全关键控制过程集合(以 CA 表示),并提炼出与 CA 相对应的所有可能导致问题的非安全控制集(以 UCA 表示),主要根据以下几条原则进行提炼:

1)未能提供所需的安全性措施;

2)产生了不安全的错误输出;

3)所需的控制措施提供的时机与预期不符(太早或太晚)。

Step3 围绕 Step2 中所提炼的 UCA 集中每一个非安全的控制过程,梳理得出其所对应的 F_i。如图 3 所示,对于每一个 UCA_i,寻求一个或多个 F_i,将当前控制过程所对应的软件自身实现逻辑问题 SW_i、软硬交互 HW_i、人机交互 PE_i、运行环境 EN_i、某种特定运行状态 OT_i 与 UCA_i 进行关联,实现每一个非安全的控制过程与若干 F_i 之间的映射。当遍历完 UCAS 集合中的所有非安全的控制过程后,即可得到所有的 F_i,将所有的 F_i 组合在一起,可以形成软件系统层面的安全约束集 *Constraints-Sets*,表示为 *Constraints-Sets* $=\{F_1, F_2, \cdots, F_n\}$。

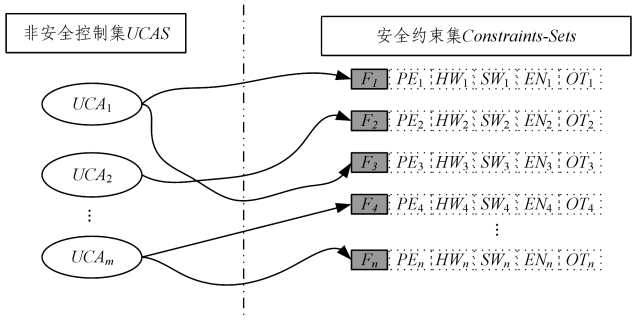

图 3 非安全控制集和安全约束集的映射关系

Fig. 3 Mapping relationship between UCAS and constraints-sets

3.2.2 安全关键软件验证集的设计

在构建完安全关键软件需求约束集后,需要依据安全约束集对安全关键软件进行验证。为此,本文提出了基于约束集的安全关键软件验证集设计方法。

遍历 *Constraints-Sets* 中的每个 F_i,对每个 F_i 进行基于测试模型的转换,生成与之对应的验证用例 TC_i。该测试模型的转换基于特定的测试工具实现,对于每个 F_i 中的 PE_i, HW_i, EN_i, SW_i, OT_i,将其转换为相应的测试条件、测试环境、测试数据、测试指令以及测试执行动作等组合,从而形成相应的验证用例 TC_i,所有的 TC_i 组合在一起,构成了安全约

束集 *Constraints-Sets* 相对应的软件安全性验证集 *TC-Sets*。

3.2.3　进行基于安全性验证集的测试验证

在完成对安全约束集 *Constraints-Sets* 及与之相对应的软件安全性验证集 *TC-Sets* 的建立后，在特定的测试环境中，以 *TC-Sets* 中的每个验证用例 TC_i 作为验证输入，自动执行，并判断验证结果是否与预期结果相符。若出现与预期结果不符的现象，将当前所采用的 TC_i 和验证结果反馈至软件需求提出方和开发人员，并进行相应的修正完善。

3.3　基于多维要素的安全关键软件验证方法的应用

3.3.1　待验证软件描述

限于篇幅，本文选用某电源管理软件的放电处理任务进行方法验证。软件运行于中心处理计算机 SMU 上。在放电任务处理中，当检测到光照/地影标志 Flag 变为地影状态时，开始通知蓄电池进行放电以维持对其他设备的能源供应（蓄电池电量百分比初始态为 Battery-Capacity），并累计放电时间 ΔT，当 ΔT 持续到一定的时间，蓄电池累计放电量 $\Delta Discharge$ 先后达到门限值 *threshold*1 和 *threshold*2（*threshold*2＞*threshold*1）时，分别给出相应的报警，说明当前蓄电池容量已经不足，由软件进行相应的报警处理。在该过程中，操作人员可以根据需要发送指令对 Battery-Capacity，并对 *threshold*1 和 *threshold*2 进行设置。

3.3.2　验证过程描述

Step1　抽象放电任务的安全性处理模型。

基于 3.2 节的分析，结合放电任务处理任务的描述，建立其安全性处理模型，如图 4 所示。可以看到，在该模型中，放电处理任务运行于 SMU 上，接收操作人员对关键参数设置指令，从电源控制器提取当前光照/地影标识来进行放电算法处理，并为其他设备进行能源供应，实时计算累计放电量，并向操作人员提供显示信息。

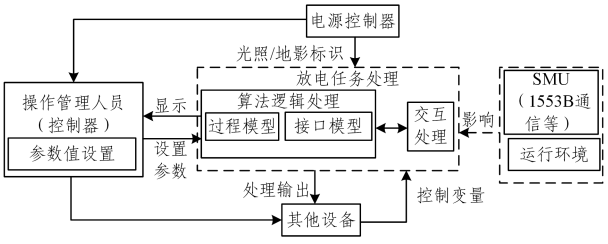

图 4　放电处理任务的控制过程处理模型

Fig. 4　Control process model for discharge task

Step2　提炼出放电处理过程中所有可能导致问题发生的非安全控制集 *UCA*。

根据 3.2 节的分析及给出的原则，可以得出，在放电处理任务中，安全关键事件是指蓄电池放电电量超限，围绕这一事件，得出两个非安全控制集 *UCA*1 和 *UCA*2：*UCA*1 表示蓄电池放电电量超限，但软件未给出相应报警和处理；*UCA*2 表示蓄电池放电电量未超限，但软件误判，进行了报警和处理。

Step3　构建放电处理过程中的安全约束集 *Constraints-Sets*。

根据 3.2 节中给出的安全约束集 *Constraints-Sets* 的构建方法，结合放电任务的处理过程，梳理得出 4 个 F_i，并分解其构成元素 PE_i，HW_i，EN_i，SW_i，OT_i，*Constraints-Sets* ＝ $\{F_1, F_2, F_3, F_4\}$，构建过程如表 1－表 4 所列。

表 1　F_1 要素模型的构成元素及其说明

Table 1　Conponent parameters and descriptions of F_1 factor model

构成元素	元素说明
PE_1	—
HW_1	*SMU* 切机或 1553*B* 通信异常
SW_1	算法未提供硬件通信异常时对 ΔT 和 $\Delta Discharge$ 进行持续累计的措施
EN_1	地影状态
OT_i	—

表 2　F_2 要素模型的构成元素及其说明

Table 2　Conponent parameters and descriptions of F_2 factor model

构成元素	元素说明
PE_2	—
HW_2	—
SW_2	未检测到 *Flag* 为地影，未放电
EN_2	当前真实状态为地影，但其标识 *Flag* 被打翻
OT_2	—

表 3　F_3 要素模型的构成元素及其说明

Table 3　Conponent parameters and descriptions of F_3 factor model

构成元素	元素说明
PE_3	设置 Battery-Capacity 不合理
HW_3	—
SW_3	使用不合理门限值参与计算
EN_3	地影状态
OT_3	—

表 4　F_4 要素模型的构成元素及其说明

Table 4　Conponent parameters and descriptions of F_4 factor model

构成元素	元素说明
PE_4	设置 *threshold*1 和 *threshold*2 不合理
HW_4	—
SW_4	使用不合理门限值参与计算
EN_4	地影状态
OT_4	—

通过上述分析，共 4 个要素结构 F_1，F_2，F_3，F_4 对放电处理安全约束集 *Constraints-Sets* 的构建产生了影响。

F_1：当前为地影状态，若 SMU 发生切机或 1553B 通信异常故障，而软件逻辑处理中未能提供对 ΔT 和 $\Delta Discharge$ 的保护动作时，无法统计蓄电池放电的真实时间和累计放电量，导致在蓄电池放电超限后，未能给出报警提示和相应处理。

F_2：当前为地影状态，但光照/地影标志被环境干扰，软件无法判断到当前处于地影状态，未能给出放电的指令通知，导致外围设备缺乏能源供应。

F_3：当前为地影状态，操作人员设置 Battery-Capacity 不合理（超过 100%），而软件逻辑中又缺乏合理性判断，导致软件累计的 ΔT 和 $\Delta Discharge$ 与蓄电池的真实状态不符。

F_4：当前为地影状态，操作人员对 *threshold*1 和 *threshold*2 的设置不合理（*threshold*1＞*threshold*2），而软件逻辑中又缺乏合理性检测，导致报警超限处理动作与预期不符。

Step4　构建放电处理过程中的安全验证集 *TC-Sets* 并进行验证。

对 Step3 中得出的 *Constraints-Sets* 中的 F_1，F_2，F_3，F_4

分别进行测试模型转换，生成与其对应的验证用例 TC_1，TC_2，TC_3 和 TC_4。

TC_1：测试环节，通过指令设计 SMU 切机或断电的故障；TC_2：测试环节，配置当前状态为地影状态，同时通过模拟器故障注入修改 *Flag* 值，使该值不标识当前为地影状态；TC_3：测试环节，通过指令将 Battery-Capacity 设置为大于 100% 的数；TC_4：测试环节，通过指令设置 *threshold*1 和 *threshold*2，使 *threshold*1＞*threshold*2。

随后，构建测试环境，如图 5 所示。图中，放电处理任务运行于中心处理计算机 SMU 上，操作人员通过控制计算机进行操控，并通过测试网向中心处理计算机发送控制指令，并接收其发送的参数信息；中心处理计算机与电源控制器以及其他设备通过 1553B 总线进行通信，由中心处理计算机通过 1553B 总线发出指令，电源控制器及其他设备通过 1553B 总线接收指令，并将自身的参数状态信息通过 1553B 总线反馈至中心处理计算机。此外，测试环境中，还配置了 1553B 故障注入模拟器，用于模拟外部故障并进行注入。

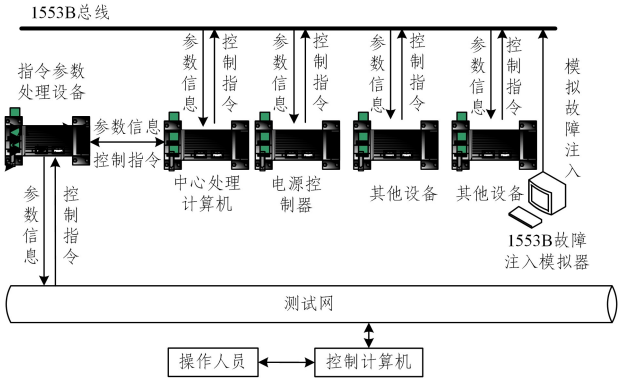

图 5 放电处理任务测试场景图

Fig. 5 Test environment for discharge task

在图 5 的测试环境中，以 Step4 中构建的 *TC-Sets* 中的每个验证用例 TC_1，TC_2，TC_3 和 TC_4 作为验证输入并自动执行，判断验证结果是否与预期结果相符。

通过测试验证得出验证结果及结论，并列出其与安全关键影响要素、验证用例之间的对应关系，如表 5 所列。

表 5 影响要素、验证用例与验证结论的对应关系

Table 5 Mapping among influencing factor, verification set and conclution

序号	影响要素	验证用例	验证结果及结论
1	F_1	T_1	硬件异常时，软件停止对 ΔT 和 $\Delta Discharge$ 的累计计算，不符合设计本意。因此，软件应为 ΔT 和 $\Delta Discharge$ 等提供重要数据保护的功能，防止硬件发生故障时，ΔT 和 $\Delta Discharge$ 丢失或不连续
2	F_2	T_2	关键状态标志发生跳变后，软件未能有效识别，产生了错误输出。因此，软件应向操作人员提供对光照/地影状态标识的设置指令接口，防止在环境干扰下，其状态发生跳变
3	F_3	T_3	对 Battery-Capacity 设置不合理的数值时，无纠错措施。因此，软件需求层面应明确操作人员对 Battery-Capacity 设置的合理值范围
4	F_4	T_4	软件进行报警超限处理的动作与预期不符。因此，在软件需求层面应明确操作人员对 *threshold*1 和 *threshold*2 设置的约束，明确 *threshold*1＜*threshold*2

验证结果分析：在对放电处理任务进行安全性验证时，基于系统性多维要素，分别从硬件异常、环境干扰、操作人员指令设置错误的角度出发，构建了 4 个放电处理过程的安全约束集，并形成与之对应的安全验证集。通过验证得出：软件在需求设计阶段，若未从系统的角度出发，提供系统性的安全性处理措施，则在硬件异常、环境干扰或操作人员指令设置错误时，将会提供不符合预期的错误输出。

随后，采用传统的验证方法对放电处理任务再次进行安全性验证，由验证结果可知，表 5 中所得出的 4 条验证结论均未能通过传统方法得出。这是由于传统方法仅关注软件自身，其进行验证用例设计时缺少对硬件环境、状态标志、人机交互等系统性验证要素的设计，无法形成表 5 中所构建的安全关键软件影响要素和对应的验证用例，因此未识别到表 5 中给出的 4 个潜在的系统性问题。

结束语 结合在实际安全关键软件研制过程中的总结和积累，文本从系统的角度出发，开展了对安全关键软件验证方法的研究。首先，对影响安全关键软件的要素进行建模，形成包含 5 个元素的影响安全关键软件的多维要素结构模型；其次，基于多维要素的安全关键软件验证方法，通过抽象关键处理任务的处理过程模型提炼其对应的非安全控制集方法，梳理得出安全关键软件需求约束集的构建方法和步骤，实现需求约束集和多维影响要素结构模型的有机整合，形成对安全关键软件验证的模型基础；最后，围绕软件安全性需求约束集进行测试模型的转换，生成了相应的安全性验证集，并在特定的测试环境中开展了相关验证工作。

相关应用表明，所提方法可以有效解决安全关键软件中许多深层次的问题，尤其是软件需求层面存在的安全设计缺陷，这些潜在问题往往难以通过常规的测试验证方法进行识别和发掘，因此本文提出的系统化的验证方法具有很好的应用效果。

在实际的工程应用中，应用本文所提方法时还需注意以下事项：

1）在应用本文方法之前，必须检查安全关键软件需求的完备性。需求遗漏是引发安全关键软件问题的主要原因之一，在软件系统层面，对软件系统级功能、系统相关的安全性需求应该事先进行完整性分析，如果描述不完整，需要由软件系统需求人员进行及时补充与完善。

2）必须检查安全关键软件安全性需求定义的无歧义性。应严格对安全关键软件需求约束集中涵盖的每条安全性影响要素模型结构进行检查，确保每条影响要素模型结构中的各个要素的定义表达清晰准确，具有唯一性。

3）当安全关键软件存在多种运行模式时，在梳理安全关键软件的非安全控制集和安全性影响要素模型结构时，还需要考虑软件当前所处的特定运行模式及其产生条件，对于两个运行模式之间切换的临界模式，需要将运行模式作为安全性影响要素之一进行定义和分析。

围绕安全关键软件的相关验证工作，后续的研究计划包括：

1）尝试开展对安全性需求约束集和安全性验证集的进一步形式化描述，并尝试融入逻辑代数操作，表征安全关键软件的运行状态，刻画其所有安全控制集合之间的转移特性，基于布尔运算等手段完成状态集合之间的逻辑运算，避免状态空间爆炸的情况。

2）针对安全关键软件，尝试引入新的可靠性评价参数，研究新的可靠性评价方法，提高面向安全关键软件的可靠性评价的针对性和评价精度。

参 考 文 献

[1] ATHALYE P, MAKSIMOVIC D, ERICKSON R, et al. High-Performance front-end converter for avionics applications[J]. IEEE Transactions. on Aerospace and Electronic Systems, 2003, 39(2): 462-470.

[2] JIANG M X. Research on Safety Testing for Airborne Software [D]. Nanjing: Nanjing University of Aeronautics and Astronautics, 2015. (in Chinese)
姜梦霞. 机载软件的安全性测试研究[D]. 南京：南京航空航天大学，2015.

[3] GJB/Z 102A－2012. 军用软件安全性设计指南[OL]. http://www.gjb.corn.cn/. 2012.

[4] GJB/Z 142－2004. 军用软件安全性分析指南[OL]. http://www.gjb.com.cn/. 2004.

[5] FAN X G, CHU W K, ZHANG F M. Surveys of software safety [J]. Computer Science, 2011, 38(5): 8-13. (in Chinese)
樊晓光，褚文奎，张凤鸣. 软件安全性研究综述[J]. 计算机科学，2011，38(5)：8-13.

[6] HUANG Z Q, XU B F, KAN S L, et al. Survey on Embedded Software Safety Analysis Standards, Methods and Tools for Airborne System[J]. Journal of Software, 2014, 25(2): 200-218. (in Chinese)
黄志球，徐丙凤，阚双龙，等. 嵌入式机载软件安全性分析标准、方法及工具研究综述[J]. 软件学报，2014，25(2)：200-218.

[7] NASA. Software Safety: NASA-STD 8719.13C(2013) [S]. NASA Technical Standard, 7-77.

[8] STRINGFELLOW M, OWENS B, LEVESON N, et al. A Safety-Driven Systems Engineering Process[J]. INCOSE International Symposium, 2008, 18(1): 605-619.

[9] STRINGFELLOW M V, LEVESON N G, OWENS B D. Safety-Driven Design for Software-Intensive Aerospace and Automotive Systems[J]. Proceedings of the IEEE, 2010, 98(4): 515-525.

[10] ISHIMATSU T, LEVESON N G, THOMAS J P, et al. Hazard Analysis of Complex Spacecraft Using Systems-Theoretic Process Analysis[J]. Journal of Spacecraft and Rockets, 2014, 51(2): 509-522.

[11] ABDULKHALEQ A, WAGNER S, LEVESON N. A Comprehensive Safety Engineering Approach for Software-Intensive Systems Based on STPA [J]. Procedia Engineering, 2015, 128(4): 2-11.

[12] KNIGHT J C, LEVESON N G. Should software engineers be licensed? [J]. Communications of the Acm, 2002, 45(11): 87-90.

[13] ABDULKHALEQ A, WAGNER S. A Software Safety Verification Method Based on System-Theoretic Process Analysis[C]// International Conference on Computer Safety, Reliability, and Security. Springer International Publishing, 2014.

[14] ABDULKHALEQ A, WAGNER S. Integrated Safety Analysis Using Systems-Theoretic Process Analysis and Software Model Checking[C]// International Conference on Computer Safety, Reliability, and Security. Springer, Cham, 2014.

[15] FLEMING C H, LEVESON N. Integrating Systems Safety into Systems Engineering during Concept Development[J]. INCOSE International Symposium, 2015, 25(1): 989-1003.