计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(8):274−280 [doi: 10.15888/j.cnki.csa.007944]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

航天嵌入式软件代码逻辑分析①

左万娟[1,2], 董　燕[1,2], 黄　晨[1,2], 王小丽[1,2]

[1](北京控制工程研究所, 北京 100190)
[2](北京轩宇信息技术有限公司, 北京 100190)
通讯作者: 左万娟, E-mail: spacecraft001@163.com

摘　要: 为提高航天嵌入式软件的测试质量、确保航天型号任务的圆满完成, 对航天嵌入式软件代码审查重要内容之一的代码逻辑分析进行了研究. 通过对软件缺陷的机理、缺陷查找过程、缺陷暴露过程、以及缺陷引发后果的分析, 结合多年软件测试工程实践经验的总结, 提出了场景分析法、时序分析法、假想故障追源法等 10 种主要的代码逻辑分析方法. 开展了代码逻辑分析方法的应用分析、代码审查与其它测试手段之间的对比分析, 通过分析, 给出了代码审查的工程适用性说明. 研究成果已在航天型号软件第三方评测中全面推广应用, 实践数据表明, 应用效果良好, 使代码审查的缺陷发现率由业界公认的 30%~70% 提升至 90% 以上. 相关分析方法和分析思路对动态测试设计以及软件缺陷自动化检测工具的研发均具有一定的参考作用.

关键词: 航天; 嵌入式; 软件; 代码逻辑分析; 代码审查

引用格式: 左万娟,董燕,黄晨,王小丽.航天嵌入式软件代码逻辑分析.计算机系统应用,2021,30(8):274–280. http://www.c-s-a.org.cn/1003-3254/7944.html

Aerospace Embedded Software Code Logic Analysis

ZUO Wan-Juan[1,2], DONG Yan[1,2], HUANG Chen[1,2], WANG Xiao-Li[1,2]

[1](Beijing Institute of Control Engineering, Beijing 100190, China)
[2](Beijing Sunwise Information Technology Ltd., Beijing 100190, China)

Abstract: In order to improve the test quality of aerospace embedded software and ensure the successful completion of aerospace tasks, we study code logic analysis, one of the important contents of code inspection for aerospace embedded software. We analyze the mechanism, finding, exposure, and consequences of software defects and summarize many years of engineering practice in software testing. On this basis, we put forward ten methods for code logic analysis, such as scene analysis, time sequence analysis, and imaginary fault source tracing. The application of code logic analysis methods are analyzed and code inspection is compared with other test methods, through which the engineering applicability of code inspection is given. The methods have been widely adopted in the third-party testing of aerospace software. Practical data show satisfying application effect as the defect detection rate of code inspection has increased from generally accepted 30%~70% to more than 90%. The methods and ideas can provide reference for the design of dynamic test and the research and development of automatic detection tools for software defects.

Key words: aerospace; embedded; software; code logic analysis; code inspection

航天嵌入式软件, 因其自主处理能力强、运行实时性要求高、故障诊断与处理手段多、在轨运行时间长以及在轨运行环境复杂等特点, 导致其部分运行场景很难在地面实现真实状态下的动态验证. 因此, 在地

① 基金项目: 国家自然科学基金 (61802017); 装备预研领域基金 (61400020407)
Foundation item: National Natural Science Foundation of China (61802017); Pre-research Project of Equipment (61400020407)
收稿时间: 2020-09-26; 修改时间: 2020-10-21; 采用时间: 2020-11-16; csa 在线出版时间: 2021-07-31

面所开展的动态测试往往会存在这样或那样的测试缺口, 导致了某些软件问题的在轨发生, 这对于有着高可信高可靠高自主要求的航天嵌入式软件而言, 是难以接受的. 同时, 随着软件定义卫星设计理念的提出, 软件规模和复杂度的不断攀升, 也导致软件潜在缺陷越来越多[1,2]. 因此, 研究软件测试方法, 进一步提升软件测试质量势在必行.

软件测试, 根据是否运行软件, 一般分为静态测试和动态测试, 其中, 针对静态测试, 一般采用基于工具的静态分析、和基于人工的代码审查等测试手段. 根据目前的工程实践, 对于有着高可信高可靠要求的航天嵌入式软件而言, 基于工具的静态分析手段仍然无法取代人工代码审查, 只能作为人工代码审查的补充手段. 因此, 在现有条件下, 对于航天嵌入式软件的质量保证而言, 人工代码审查仍然是不可或缺的一种重要技术手段.

自从 1976 年首次提出代码审查 (code inspection) 以来, 代码审查一直被认为是一种重要而且有效的改进软件质量的方法[3,4]. 经验表明, 组织良好的代码审查, 可以发现程序中 30%~70% 的编码和逻辑设计错误[5]. 高质量的代码审查能够发现 60% 以上的软件问题, 而且往往能够发现很多通过工具和动态测试无法发现的深层次代码问题, 如算法实现问题、软件架构问题、时序逻辑问题等[6]. 代码审查可以比动态测试更有效地发现某些特定类型的缺陷, 且实施时无需特别条件, 成本较低[7]. 众多研究成果表明, 相比动态测试, 代码审查更为高效、灵活且发现问题能力强.

但是, 由于代码审查本质上仍然属于一种既费时又费力的技术手段, 且测试结果严重依赖于人的能力, 因此, 虽然是目前工程实践中的一种主要技术手段、但是针对性的研究并不多. 当前软件测试的研究热点更多地集中在工具、自动化和人工智能等方面[8–12]. 有限的针对基于人工代码审查所开展的研究多着眼于检查项 (即检查内容) 的研究与总结[13–16], 对方法的研究与总结则很少, 尤其在基于人工的代码逻辑分析方面, 尚无相关研究成果发表. 而基于检查项的代码审查方法, 属于片段式检查, 缺乏对代码的整体逻辑分析, 不利于发现代码中潜藏的深层次复杂逻辑问题.

本文针对代码审查重点内容之一的代码逻辑分析开展研究, 提出了场景分析法、时序分析法、假想故障追源法等 10 种主要的代码逻辑分析方法, 并开展了工程应用分析.

1 分析方法

“分析”, 通俗来说, 是以某种方式将复杂对象分解为更小的部分, 以更好地理解该对象的过程[17].

通过对近 10 年来航天型号软件在轨、在研以及第三方评测中软件缺陷的机理、缺陷查找过程、缺陷暴露过程以及缺陷引发后果的研究, 提出了场景分析法、时序分析法、假想故障追源法、答疑解惑分析法等十种主要的代码逻辑分析方法, 分别予以阐述.

1.1 场景分析法

软件系统的执行过程可看作是大量场景的有序序列[18].

针对特定的软件功能, 结合软件的运行时序及运行状态, 构造不同的软件运行场景, 进行针对特定场景的软件运行状态与逻辑分析.

通过此法, 重点确认针对各种可能的运行场景, 软件是否均能达到设计预期; 针对特定场景, 是否存在设计失效、甚至进而引发严重的不良后果.

实例说明如下:

某单机采用双机冷备份, 设计了如下自主故障诊断及处理策略:

如果诊断到主份连续故障 20 s, 则临时切换至备份 (主份断电、备份加电), 2 s 后再切换回主份 (主份加电、备份断电).

切换回主份后, 如果诊断到主份设备仍然故障, 则当连续故障达到 100 s 则永久切换至备份.

针对上述设计, 需要构造如下场景, 并对相关代码逻辑进行分析:

场景 1. 临时主备切换之后, 主份恢复正常 (默认备份正常).

场景 2. 临时主备切换之后, 主份仍然故障 (默认备份正常).

经场景 2 分析发现, 由于主备故障诊断共用了一个连续故障计时变量, 当主份故障 20 s 临时切换至备份期间, 由于备份正常, 导致连续故障计时变量被清零, 则切换回主份后, 20 s 后会再次执行临时切换备份处理. 即, 在场景 2 下, 软件会周期性地执行主备切换, 这显然违背了设计预期. 按照设计预期, 如果主份持续故障, 则 100 s 后应执行永久切备份处理.

上述设计缺陷, 如果仅做场景一分析, 是无法检出

的; 如果分析过程中, 未考虑备份状态对软件运行状态的影响, 也无法检出.

1.2 时序分析法

航天器实际上是一个分布的、网络化的嵌入式系统, 在这个系统中各个单机采用相同或不同的中央处理单元 (CPU) 及编程语言, 彼此之间通过各种串口、并口或总线, 采用中断或查询方式进行通讯[19], 这就要求彼此之间的通讯时序必须匹配良好, 否则将导致通讯失败, 影响航天器的正常运行.

时序分析法, 重点针对航天器系统中各个单机之间的通讯时序匹配性开展分析. 至于单机内部的运行时序, 一般可以通过变量分析以及代码逻辑分析中的场景分析法等来确认.

实例说明如下:

软件通过 422 串口与上位机通讯, 接收上位机指令, 并回复应答数据.

为确认软件设计是否满足上述通讯需求, 应重点做如下时序分析:

(1) 接收时序分析: 接收的及时性必须保证, 否则导致接收丢字符.

(2) 应答时序分析: 应答的及时性必须保证, 否则影响上位机软件接收应答数据.

根据上述分析, 代码审查时重点关注如下时序相关设计:

(1) 串口中断的处理时间: 如果一个字符的接收/发送触发一次串口中断, 则串口中断的设计必须简捷, 即, 中断处理时间不能过长, 否则, 将不能及时接收/发送下一个字符.

(2) 高优先级中断的处理时间: 如果有比串口中断更高优先级的中断, 则高优先级中断的处理时间必须严格控制, 避免影响串口中断响应及处理的及时性.

(3) 关中断时间: 如果软件中采取了关中断保护设计, 则关中断的时间必须严格控制, 避免影响串口中断响应及处理的及时性.

1.3 假想故障追源法

针对软件中的某些特定设计, 需要建立假想故障, 并追踪溯源, 分析代码设计是否存在触发假想故障的源头, 如果存在, 则为潜在设计缺陷.

以常见的 while 循环设计为例, 建立假想故障为 while 死循环, 分析代码设计上是否存在触发 while 死循环故障的源头. 此类缺陷较多, 实例说明如下:

实例 1. 当软件依赖于特定的硬件状态条件退出 while 循环时, 如果硬件状态始终不具备, 则导致 while 死循环.

实例 2. 依赖于计数条件退出 while 循环时, 如果计数条件被破坏导致计数无法累加, 则因计数条件始终不满足而导致 while 死循环.

1.4 答疑解惑分析法

代码中总是会或多或少地存在一些令人困惑的设计细节, 这通常是由于需求描述的颗粒度限制所导致的, 这些令人困惑的设计细节就是答疑解惑分析法的分析对象, 分析目标就是实现对这些设计细节的答疑解惑, 即, 理清代码设计的理由和依据, 以确认代码设计的正确性. 如果通过分析无法解除疑惑, 则通常在这些设计细节中是存在潜在的设计缺陷的.

实例说明如下:

软件每秒给下位机软件发同步校时指令, 针对指令中的同步校时时间 *Time.sync*, 代码设计如下:

Time.sync = *Time.Star_Orbit* – 0.02 + (*Time.DeltaT* * 1.0e–6);

针对同步校时时间 *Time.sync*, 需求中并未明确说明其计算方法. 通过代码分析发现, *Time.Star_Orbit* 是当前控制周期起始时刻, 操作“*Time.Star_Orbit*–0.02”, 将时间校正到上一控制周期 TCTM 任务起始时刻. 而操作“+ (*Time.DeltaT* * 1.0e–6)”中, 变量 *Time.DeltaT* 是上一次 (也就是上一秒) 同步校时指令发出时刻相对于指令发出所在控制周期的 TCTM 任务起始时刻的时间偏差. 综上, 这是一个很令人困惑的设计, 困惑点如下:

(1) 参与 *Time.sync* 计算的 *Time.Star_Orbit*–0.02 与 *Time.DeltaT* * 1.0e–6 的时间基准不同, 导致计算结果的物理意义不明确.

(2) 为什么不是将 *Time.sync* 调整到同步校时指令发出时刻呢?

经与开发方沟通反馈, 代码修改如下:

Time.sync = *Time.Star_Orbit* + 0.65;

修改后的代码中, 0.65 是根据任务调度时序计算得到的同步校时指令发出时刻相对于当前控制周期起始时刻的时间偏差, 即, *Time.sync* 为同步校时指令发出时刻. 至此, 疑惑解除, 分析通过.

1.5 类似设计对比法

代码中经常会存在一些类似设计, 比如, 针对同类多个单机的操作、不同场景下的同类处理, 等. 针对此

类设计, 开发人员通常采用代码克隆的方式实现.

所谓代码克隆, 是指软件开发中由于复制、粘贴引起的重复代码现象. 研究指出, 一般商业软件中存在 5%~20% 的重复代码[20].

针对代码中的类似设计, 可采用类似设计对比法开展分析. 即, 针对类似设计代码进行比对, 查看类似设计之间是否存在差异, 分析差异之处的设计正确性.

通常, 通过简单的比对, 可以找出因代码克隆时的疏忽而导致的设计缺陷. 但是, 也有些较深层次的设计缺陷, 需要通过更审慎的分析才可发现.

实例说明如下:

软件在 task4 中根据标志 *hk6start* 启动 HK6 相关处理, 正常结束 HK6 处理时, 会清标志 *hk6start*、并发 TC_HK6 指令停止 HK6 处理. 但是, 软件在处理某故障时存在仅清标志 *hk6start*、未发 TC_HK6 指令的设计, 与正常结束 HK6 处理的流程不一致.

类似设计对比法的应用, 主要有如下两个方面:

(1) 通过比对发现类似设计上的差异, 通过差异分析查找缺陷.

(2) 某处类似设计中检出缺陷后, 应继续分析其它类似设计处是否存在类似缺陷.

1.6 相关性 (影响域) 分析法

有些情况下, 软件的各个功能之间存在一定的相关性, 即, 某一功能的实现方式可能会影响另一功能的实现结果. 因此, 需要开展相关性 (影响域) 分析.

实例说明如下:

需求要求以变量形式定义某 S 存储区的地址空间, 以便在轨重新分配地址空间.

软件实现时, 以变量形式定义了 S 存储区的地址空间, 并在实施 S 区存储时以变量形式访问地址, 满足需求. 但是, 在 S 存储区数据下卸指令处理中, 软件按照初始设置的 S 存储区的起始/结束 (固定) 地址来进行指令参数合法性判断, 显然, 该合法性判断设计与“在轨重新分配 S 区地址空间”的需求是不符的.

1.7 星地控制匹配分析法

卫星在轨运行具有高度自主性, 同时以遥测的形式, 将其在轨运行状态打包发回地面, 供地面监控, 必要时, 地面通过遥控指令等方式对卫星实施控制. 但是, 从在轨数据打包、到数据发回地面、至地面完成解析与数据判读, 总是会有或多或少的时延, 这也就导致地面可能在并未实时获知卫星在轨运行状态的情况下对卫星进行了遥控控制, 从而可能导致卫星自主控制与地面遥控控制之间发生相互间的干扰、覆盖等不匹配行为.

星地控制匹配分析法, 针对星地控制的匹配性开展分析, 其分析对象一般是星上自主和地面遥控两种手段均可控制的功能, 分析两种控制手段之间是否存在相互干扰、覆盖等不匹配行为并造成了不良后果.

目前, 针对同时具备星地两种控制手段的功能, 通常的设计准则是地面优先, 即, 地面遥控可以覆盖星上自主控制, 反之不行. 但是, 也存在个例. 比如模式转换, 如果在星上自主转模式 M 之后, 地面再次发遥控指令转模式 M, 则星上软件一般应屏蔽地面发的遥控转模式 M 指令, 否则, 转模式时的初始化操作会干扰星上正常的自主模式控制过程.

实例说明如下:

某部件控制指令, 有星上自主发出和地面遥控发出两种手段.

软件在 task1 中根据当前运行状态自主设定控制参数、并发送该指令.

软件在 task2 中接收地面遥控指令, 其中的控制参数由地面设定.

根据上述设计, 软件形成如下运行时序: task2 接收遥控指令→; task1 自主设置指令→; task1 发送指令. 可见, 遥控指令必定被自主指令所覆盖, 从而导致遥控失效.

1.8 接口软件状态分析法

有些情况下, 被测软件会采集与其接口的软件状态, 并将该状态作为特定处理的判定条件, 即, 接口软件的状态会影响被测软件的运行状态, 而这通常是软件设计师容易忽视的地方. 测试过程中, 忽略接口软件状态对被测软件状态的影响, 或者对接口软件状态未仔细探究而采取了想当然的心态, 都容易导致被测软件相关设计缺陷被遗漏.

当接口软件状态影响到被测软件的运行状态时, 需要分析各种可能的接口软件状态对被测软件的影响, 以确认被测软件在相关设计上的正确性.

实例说明如下:

某故障处理策略: 陀螺连续故障计数达到 100, 则给陀螺断电 10 个控制周期, 然后再给陀螺加电.

但是, 由于陀螺断电期间, 被测软件采集到的陀螺通讯状态为异常, 导致连续故障计数不变 (注: 通讯正

常情况下, 才进行诊断并操作连续故障计数), 从而导致软件重复执行连续故障计数 100 的处理分支, 与设计预期不符.

1.9 隐含需求分析法

隐含需求通常指那些必须要实现、但是需求分析人员又没有意识到需要把它们在需求规格说明中清晰描述出来的需求. 这些需求, 一旦未予实现, 通常会引发一定的不良后果.

实例说明如下:

故障诊断及处理功能中, 如果采取了故障后给设备断电、再加电的处理策略, 则需要考虑持续故障情况下, 是否会导致设备重复断电、加电, 这种持续故障情况下的设备重复断电加电通常是设计上所不期望的. 但是, 通常的需求规格说明中仅会提出故障后给设备断电再加电的需求, 而不会明确说明持续故障后的处理需求. 因此, 持续故障情况下的处理策略就属于隐含需求, 需要明确需求后对相关处理开展分析检查.

隐含需求通常涉及如下方面:

(1) 遵循的标准、芯片手册: 比如, 标准及芯片手册中的操作规范、时序要求等.

(2) 软件接口交互需求: 比如, 交互双方的接收与应答响应等.

(3) 硬件接口交互需求: 比如, 查询、等待硬件状态的延时涉及等.

(4) 可靠性安全性需求: 比如, 指令合法性校验、重要数据三取二等.

(5) 功能级需求: 最为复杂的一类隐含需求, 需要结合具体功能做具体分析.

1.10 需求/设计合理性分析法

在代码审查过程中, 针对需求和设计, 都需要做合理性分析, 避免盲从.

有许多需求/设计不合理的实例:

实例 1. 参数 (变量) 有初值, 同时支持上注修改. 但是, 在上注异常时软件将参数清零、而非恢复为初值或保留之前的上注值. ——设计不合理.

实例 2. 协议规定遥测中的 CAN 重新初始化标志占用 1 bit, A、B 总线共用, 发生一次 CANA 或 CANB 重新初始化, 则遥测位翻转一次. 如果在 1 个遥测轮询周期内 A、B 总线均重新初始化, 则遥测结果与 A、B 总线均未重新初始化的遥测结果是一样的, 导致遥测设计失效. ——需求不合理.

实例 3. AD 采集过程中, 如果软件采用了连续采集多次取均值的设计, 则连续采集过程中, 每次采集均应重新启动 AD 转换, 否则会导致连续采集设计无意义. ——设计不合理.

2 应用分析

2.1 应用说明

作为代码审查的重点内容之一, 代码逻辑分析侧重于功能级的代码逻辑的分析与确认. 工程应用中, 应针对代码逻辑分析与检查单、变量分析[21,22]、中断访问冲突分析[23,24]等其它代码审查重点内容开展综合性分析与确认, 以期达到最佳的工程应用效果.

2.2 应用数据分析

根据中国空间技术研究院软件产品保证中心的统计数据, 近半年内, 开展代码逻辑分析所检测出的缺陷占比为 23%.

选取近期结束的 5 个项目, 对综合开展代码逻辑分析、检查单确认、变量分析、中断访问冲突分析后的代码审查发现缺陷占比进行统计, 数据如表 1.

表 1 综合应用数据统计

序号	规模(行)	代码审查缺陷数	动态测试补充缺陷数	代码审查发现缺陷占比(%)
1	46775	41	0	100
2	7417	13	1	92.86
3	59776	61	1	98.39
4	4010	21	0	100
5	21273	39	4	90.70

数据来源说明:

(1) 统计数据取自中国空间技术研究院软件产保中心第三方评测数据库.

(2) 所选项目均为星载嵌入式软件第三方评测项目、且提交第三方评测之前均已完成开发方自测试.

(3) 仅统计已做修改程序处理的缺陷数.

(4) 统计数据不含注释错误、多余物等低级缺陷.

通过应用数据可以看出, 综合开展代码逻辑分析、检查单确认、变量分析、中断访问冲突分析后, 代码审查发现缺陷占比明显高于业界公认的 30~70%, 应用效果良好.

2.3 应用意义分析

依托具体方法, 开展综合性代码审查, 产生的意义如下:

(1) 使代码审查过程有法可依、有章可循, 有助于提升人的能力, 缩小人与人之间测试质量的差异, 实现整体测试能力的提升.

(2) 为代码设计提供参考, 编码阶段借鉴这些方法, 有助于及早规避软件缺陷, 实现缺陷早期预防.

(3) 为测试设计提供参考, 有利于提高动态测试的充分性.

(4) 为缺陷自动化检测工具研发提供参考, 有助于促进工具能力的提升.

(5) 为缺陷自动化检测工具的推广应用提供了参照目标, 工具的能力必须超越人, 才能真正取代人.

2.4 不同测试手段的对比分析

结合多年的软件测试工程实践, 开展了静态分析工具、人工代码审查、动态测试手段的对比分析如表 2.

表 2 不同测试手段对比分析

序号	因素	静态分析工具	人工代码审查	动态测试
1	测试质量的可控性	好 有工具分析数据可查, 测试质量可控性好	差 没有实际运行结果, 缺乏过程控制手段, 测试质量可控性差	好 有实际运行结果, 有过程控制手段(需求和测试设计评审、覆盖率分析等), 测试质量可控性好
2	测试质量可达性	可达上限低 仅在既有检查规则及算法范围内才能有效发现软件缺陷、且存在误报和漏报不具备复杂的时序/逻辑分析能力、不具备文实一致性检查能力	可达上限高 高级测试人员的缺陷检出率可达95%以上	可达上限中 受测试环境的支持性、测试需求分析和测试用例设计的正确性和完备性的限制
3	测试人员能力影响	小 仅需要测试人员对工具报告结果进行人工确认	大 高级测试人员与低级测试人员的测试质量差异巨大	中 测试人员能力直接影响到测试需求分析及测试设计的充分性和正确性
4	时序分析与验证能力	— 基本不具备该能力	强 便于开展各类特定的时序分析	弱 针对随机性很强的中断访问冲突问题, 缺乏解决方案针对外部干预与软件自主运行之间的特定时序组合的测试, 缺乏有效的解决方案
5	复杂逻辑/状态分析能力	— 基本不具备该能力	强 便于开展各类复杂逻辑/状态分析, 可以发现一些深层次复杂逻辑/状态问题	弱 测试环境支持能力有限, 部分复杂逻辑/状态测试, 很难实现
6	缺陷定位	提供缺陷定位信息, 需要人工确认	直接定位	无缺陷定位信息 用例执行失败后要进行软件缺陷的反向定位
7	大规模(百万行以上)软件适用性	适用	不适用	适用
8	最大的劣势	受所采用技术的限制, 只能针对部分缺陷开展分析检测, 可达上限低	测试质量严重依赖于测试人员的能力	综合效率低、耗费资源多
9	最大的优势	成本低、易实现、自动化程度高、检测速度快	测试质量可达上限高 可以发现潜在的复杂逻辑缺陷	有实际运行数据, 测试结果的准确率高

综上, 各种测试手段各有千秋, 在工程上, 尤其是对于高可信高可靠航天嵌入式软件而言, 应该多种手段并举, 充分发挥各自的优势, 形成优势互补, 共同促进软件质量的提升.

2.5 工程适用性说明

综合以上分析结果, 总结代码审查的工程适用性如下:

(1) 在测试环境受限、测试工期有限等不适宜开展动态测试的情况下, 由中-高级测试人员开展代码审查. 比如, 型号软件的初样研制阶段.

(2) 在具有高可信高可靠要求、且规模允许的软件测试中, 必须开展代码审查.

3 结束语

随着自动化测试以及工具检测相关研究的热度的

提升, 基于人工的代码审查技术的研究及其在工程实践中的推广应用都陷入了瓶颈. 但是, 在当前自动化测试以及工具检测能力都不足以保证航天嵌入式软件质量的前提下, 研究并应用人工代码审查方法仍然是必要的. 而且, 无论是工具的研发, 还是工具能力的提升, 都需要一定的出发点和参照点, 而人的分析方法、分析范围以及分析能力的提升, 都为工具研发及能力提升提供了一个很好的参照, 这也是当前条件下, 坚持人工代码审查方法研究及方法推广的意义所在.

随着软件在航天器中的作用和地位越来越突出, 软件的可信性直接关系到航天任务的成败, 不存在一种单独的方法能够完全保证软件可信[19]. NASA 从多年的软件工程中吸取的一个重要教训是没有一个单一的解决方法能够解决所有问题[25]. 因此, 为满足航天嵌入式软件的可信性要求, 需要广大的工程技术人员在软件研制各阶段综合采用各种技术和方法. 只有这样, 才能将各种技术和方法的研究与应用不断地推向新的高度.

本文的研究成果对于软件研制、动态测试、测试工具研发均有一定的参考价值. 后续将重点研究如何将研究成果推广至工具研发, 以期最大限度地提升缺陷自动化检测能力, 促进多种测试手段的互补与并举.

参考文献

1 Koru AG, Zhang DS, El Emam K, *et al*. An investigation into the functional form of the size -defect relationship for software modules. IEEE Transactions on Software Engineering, 2009, 35(2): 293–304. [doi: 10.1109/TSE.2008.90]

2 Cataldo M, Mockus A, Roberts JA, *et al*. Software dependencies, work dependencies, and their impact on failures. IEEE Transactions on Software Engineering, 2009, 35(6): 864–878. [doi: 10.1109/TSE.2009.42]

3 Fagan ME. Design and code inspections to reduce errors in program development. IBM Systems Journal, 1976, 15(3): 182–211. [doi: 10.1147/sj.153.0182]

4 Fagan ME. Advances in software inspections. IEEE Transactions on Software Engineering, 1986, SE-12(7): 744–751. [doi: 10.1109/TSE.1986.6312976]

5 王露, 庄青. 源代码自动分析、评审与测试的创新应用. 信息化研究, 2017, 43(5): 6–12.

6 秦浩. 航空军工产品配套软件定型测评. 军民两用技术与产品, 2017, (9): 53–56. [doi: 10.3969/j.issn.1009-8119.2017.09.031]

7 张建飞. 基于航空领域嵌入式软件代码审查的研究. 科技创新与生产力, 2015, (10): 108–109. [doi: 10.3969/j.issn.1674-9146.2015.10.108]

8 许振波. C 程序静态分析与错误检测 [博士学位论文]. 合肥: 中国科学技术大学, 2015.

9 高猛, 滕俊元, 陈睿, 等. 航天器软件典型缺陷模式的自动检测技术. 空间控制技术与应用, 2019, 45(5): 72–78. [doi: 10.3969/j.issn.1674-1579.2019.05.011]

10 陈睿, 杨孟飞, 郭向英. 基于变量访问序模式的中断数据竞争检测方法. 软件学报, 2016, 27(3): 547–561. [doi: 10.13328/j.cnki.jos.004980]

11 丁永康. 基于遗传算法的基本路径测试用例自动生成算法研究 [硕士学位论文]. 武汉: 华中科技大学, 2016.

12 王小银, 王曙燕, 孙家泽. 基于蚁群算法的三三组合测试用例集的生成. 计算机应用研究, 2015, 32(11): 3328–3331. [doi: 10.3969/j.issn.1001-3695.2015.11.029]

13 司艳艳. 基于嵌入式软件代码审查的研究. 科技经济市场, 2013, (6): 6–8. [doi: 10.3969/j.issn.1009-3788.2013.06.002]

14 周景科. 专业领域软件的代码审查方法研究. 电子产品可靠性与环境试验, 2016, 34(3): 39–44. [doi: 10.3969/j.issn.1672-5468.2016.03.008]

15 李庆楠. 特型软件代码审查方法在军用嵌入式软件领域应用研究. 信息与电脑, 2016, (24): 160–163. [doi: 10.3969/j.issn.1003-9767.2016.24.060]

16 徐白, 余慧敏, 周楷林, 等. 嵌入式弹载软件代码审查方法研究. 电脑知识与技术, 2018, 14(10): 240–241.

17 梅宏, 王千祥, 张路, 等. 软件分析技术进展. 计算机学报, 2009, 32(9): 1697–1710.

18 张卫祥, 张敏, 窦朝晖, 等. 基于过程与场景分析的航天应用软件测试方法. 测控技术, 2020, 39(1): 30–35.

19 杨孟飞, 顾斌, 郭向英, 等. 航天嵌入式软件可信性保障技术及应用研究. 中国科学: 技术科学, 2015, 45(2): 198–203.

20 Binkley D. Source code analysis: A road map. https://doi.ieeecomputersociety.org/10.1109/FOSE.2007.27s, (2007-04-24).

21 左万娟, 虞砺琨, 黄晨, 等. 基于变量操作特征分析的软件缺陷模式研究. 航天控制, 2018, 36(5): 64–69.

22 左万娟, 虞砺琨, 王小丽, 等. 关联性变量分析在软件测试中的应用研究. “测试性与智能测控技术”——2018 年中国航空测控技术专刊. 珠海, 中国. 2018. 142–145.

23 黄晨, 董燕, 王小丽, 等. 基于表格的中断数据冲突静态分析方法. 空间控制技术与应用, 2016, 42(5): 57–62. [doi: 10.3969/j.issn.1674-1579.2016.05.011]

24 董燕, 黄晨, 王小丽, 等. 基于参数类型和访问序的数据冲突静态分析方法. 空间控制技术与应用, 2018, 44(6): 62–68. [doi: 10.3969/j.issn.1674-1579.2018.06.010]

25 NASA. NASA-GB-001-96 Software program Software management guidebook. US: NASA, 1996.