中图分类号：TP311.5

论 文 编 号 ：10006BY1006121

博 士 学 位 论 文

DOCTORAL DISSERTATION

面向方面的实时性需求建模及需求评审支撑技术研究

作 者 姓 名　张 辉 辉

学 科 专 业　计算机软件与理论

指 导 教 师　刘 超　教 授

Tao Yue (Simula Research Laboratory)

培 养 院 系　计算机学院

2017 年 04 月

Research on Aspect-Oriented Real-Time Requirements Modeling and Supporting Techniques for Requirements Inspection

A Dissertation Submitted for the Degree of Doctor of Philosophy

Candidate：Huihui Zhang

Supervisors：Prof. Chao Liu

Tao Yue (Simula Research Laboratory)

School of Computer Science & Engineering

Beihang University, Beijing, China

中图分类号：TP311.5
论文编号：10006BY1006121

博 士 学 位 论 文

面向方面的实时性需求建模及需求评审支撑技术研究

作者姓名　张辉辉　　申请学位级别　工学博士

导师姓名　刘超，Tao Yue　　职称　教授，Simula 首席研究员

学科专业　计算机软件与理论　　研究方向　软件工程

学习时间自　2010 年 09 月　日　起　至 2017 年 06　月　日止

论文提交日期　2017 年 4 月 10 日　　论文答辩日期　2017 年　月　日

学位授予单位 北京航空航天大学　　学位授予日期　　年　月　日

关于学位论文的独创性声明

本人郑重声明：所呈交的论文是本人在导师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其他人已经发表或撰写的研究成果，也不包含本人或他人为获得北京航空航天大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中作出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名：　　日期：　　年　月　日

学位论文使用授权书

本人完全同意北京航空航天大学有权使用本学位论文(包括但不限于其印刷版和电子版)，使用方式包括但不限于：保留学位论文，按规定向国家有关部门(机构)送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其他复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名：　　日期：　　年　月　日

指导教师签名：　　日期：　　年　月　日

摘　要

复杂实时系统为我们的日常生活提供不同的基础服务，它们广泛应用于不同的领域，如航空、通讯、能源、健康医疗等。在实时系统的设计中，实时相关的属性是一类关键的非功能需求。从实时系统的开发成本、产品质量、生产效率的角度出发，在需求规约和分析阶段，对这些实时相关的属性进行建模和验证，对于实时系统的成功开发是非常重要的。因此，如何系统化的捕获并规约实时系统中复杂的功能需求和各种严苛的实时约束是实时系统需求开发的关键环节。用况建模(包含用况图和用况规约)作为一种对系统需求进行捕获、规约的有效技术，已被广泛应用于工业实践中。然而，面对复杂实时系统的用况建模，各种各样的关注点(如资源约束、实时约束)横切在不同的功能需求中，导致用况图的杂乱和用况规约片段的冗余，严重影响用况模型的可重用、可维护，并降低了用况建模的效率和质量。

需求评审是识别需求缺陷的一种有效技术。不同的评审技术，如基于检查列表的技术、基于缺陷的技术，被广泛应用于工业实践中。随着用况建模作为一种有效的需求规约技术被广泛应用于工业实践中，如何进行成本-效益的用况评审是实践中的一个重要挑战。然而，目前的研究并没有给出一种系统化的变异分析技术对不同的用况评审方法进行客观公正的分析比较，从而可以进行可靠的实践推荐。由用况模型产生的用况场景通常是进行用况评审、用况驱动的测试以及其他后续活动的依据。然而，在评审资源(时间、人力)受限下，对那些复杂系统的用况模型生成的所有用况场景进行评审是一个实践挑战，尤其是复杂实时系统。因为这样的用况评审活动通常都是由领域专家进行的手工评审。 因此，需要一种自动化的用况场景选择方法，为成本-效益的用况评审提供支撑。

为了解决上述工业实践挑战，本文从实时系统的用况建模和用况评审支撑的角度展开相应的研究，提出了一系列相应的方法，选用了四个典型的工业案例和十个文献案例对这些方法进行有效性验证和分析。

1) 提出一种基于 RUCM 的实时系统用况建模方法

针对实时系统的需求建模，本文通过扩展 RUCM(Restriced Use Case Modeling)用况建模方法，提出了一种实时系统的用况建模方法 RUCM4RT，辅助需求人员捕获、规约实时系统复杂的功能需求，同时对实时特性需求进行用况建模和用况规约。RUCM4RT 是基于 UML 外廓 MARTE (UML profile for Modeling and Analysis of Real-Time and

Embedded Systems)设计的。RUCM4RT 采用元模型的形式化机制 UCMeta4RT 对创建的用况模型进行自动形式化处理。本文采用了两个案例对 RUCM4RT 进行验证，使用 RUCM4RT 成功地从选取的 40 个用况中，建模并规约了 27 个实时用况，118 个实时约束和 47 个其他非功能约束。实验结果显示，RUCM4RT 能够成功的捕获和规约系统的实时属性(如实时约束、资源约束)。

2) 提出一种面向方面的实时系统用况建模方法

针对复杂实时系统用况建模中的横切关注点问题，本文提出了一种面向方面的实时系统用况建模方法 rtAspectRUCM，支持对横切关注点(如实时约束、资源约束)的独立建模，并设计相应的编织算法实现“方面”用况模型与基用况模型的编织融合。本文分别从通讯领域、航海领域、航空领域选用三个典型的工业案例对 rtAspectRUCM 进行验证分析，比较使用 rtAspectRUCM 和不使用 rtAspectRUCM 进行横切关注点建模的功效。实验结果显示，针对横切关注点的用况建模，使用 rtAspectRUCM 平均可以节省 80% 的建模工作量。

3) 提出一种基于工业标准的用况分析方法

为了系统化的进行用况缺陷注入，以支持对不同的用况评审技术进行客观公正的有效分析，本文提出了一种基于工业标准的用况分析方法 MuRUCM。首先，依据工业标准 IEEE Std. 830-1998 全面准确的定义了 104 种用况缺陷类别。然后，依据工业标准 IEC 61882: Hazard and Operability Study (HAZOP) 创建了 219 个用况变异算子。最后，设计了一组指导规则用以辅助创建具体的缺陷生成策略。 本文选用了两个工业案例对 MuRUCM 方法进行验证分析，实验结果显示： a)：本文提出的用况缺陷分类相比较其他的文献研究，更全面综合，每一种类型的定义也更加明确。b) 创建的用况变异算子可以很容易的适用于 RUCM 用况模型，并能有效的创建不同类型的缺陷实例用以区分不同的 RUCM 覆盖策略。

4) 提出一种基于相似度函数和搜索算法的用况场景选择方法

针对评审资源(时间、人力)受限下的用况场景的择优选择问题，本文提出一种基于相似度函数和搜索算法的用况场景选择方法 S^3RUCM，用以选择那些最不相似的用况场景从而支持成本-效益的用况评审。本文选用了一个工业案例和六个文献案例，通过实证研究(Empirical Study)对 S^3RUCM 进行全面综合的验证，设计了一系列实验对选用的四种搜索算法(SSGA、(1+1) EA、AVM、RS)八种相似度计算函数(CNT、JAC、GOW、SOK、NLCS、LEV、NW、SW)进行性能分析。实验结果显示，(1+1) EA 搜索算法和 NW 相似

函数组合的性能要比其他 31 种组合显著得好，它通过选择 50%的用况场景可以发现全部注入的需求缺陷。

关键词：实时性需求建模；面向方面的用况建模；用况变异分析；用况场景选择

Abstract

Complex real-time systems provide infrastructure services for daily life and they are widely employed in different doamins, e.g., avionics, communicatin, enery, healthcare, etc. Time-related properties are a critical type of extra-functional requirements for designing real-time systems. Modeling and validating time-related properties at the requirements specification and analysis phases is important for the successful development of real-time systems in terms of cost, quality and productivity. Therefore, how to systematically capture and specify complex function requiremetns along with various time-realted requirements is the key step of requirements development for real-time systems. Use case diagrams together with use case specifications are commonly used to specify system requirements. However, when crosscutting concerns are modeled together with non-crosscutting concerns as use case models, which is especially true for complex real-time systems, use case models often face cluttered diagrams and redundant information in use case specifications. Therefore, the overall reusability of the use case models is usually low.

Requirements inspection is a well-known method for detecting defects. Various defect detection techniques for requirements inspection have been widely applied in practice such as checklist and defect-based techniques. Use case modeling is a widely-accepted requirements specification method in practice; therefore, inspecting defects in use case models in a cost-effective manner is an important challenge. However, it does not exist a systematic mutation analysis approach for evaluating inspection techniques for use case models. Use case scenarios of use case models are input elements for requirements inspection and analysis, requirements-based testing, and other downstream activities. It is, however, a practical challenge to inspect all use case scenarios that can be obtained from any non-trivial use case model, as such an inspection activity is often performed manually by domain experts. Therefore, it is needed to propose an automated solution for selecting a subset of use case scenarios with the ultimate aim of enabling cost-effective use case inspection, analysis, and other relevant activities.

To tackle the above challenge, this thesis presents a set of approaches from the perspectives of use case modeling for real-time systems and facilitating the use case inspection, respectively, which have been evaluated by four real world case studies and ten case tudies from the literature.

1) A Restricted Natural Language Based Use Case Modeling Methodology for Real-Time Systems

This thesis presents a restricted, natural language based, use case modeling methodology (named as RUCM4RT) to specify functional requirements of real-time systems as use case models, along with associated time-related constraints. RUCM4RT was proposed based on the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). In addition, in this thesis, we also propose a metamodel-based formalization mechanism named as UCMeta4RT to automatically formalize use case models. We have conducted two real-world case studies to evaluate our solution and 40 use cases were modeled, among which 27 real-time use cases, 118 time-related constraints and 47 other extra-functional (also commonly called non-functional) constraints were specified. Results show that RUCM4RT was able to handle all the real-time related elements (e.g., time-related constraints) of the use case models.

2) A Use Case Modeling Methodology for Specifying Crosscutting Concerns of Real-Time Systems

To enhance the reusability of use case models, specifically for complex real-time systems, this thesis presents an aspect-oriented use case modeling approach, named as rtAspectRUCM, for modeling crosscutting concerns, along with a weaving algortihm to automatically weave aspect use case models into their corresponding base model to facilitate, e.g., automated requirements analysis. The extended approach has been evaluated with three real world applications from communication, maritime, and aviation domains. We compared the modeling effort required to model three sets of crosscutting concerns from the real world applications, when using and not using rtAspectRUCM approach. Results showed that on average more than 80% of modeling effort for the real-world applications was saved.

3) Standard-Based Mutation Analysis for Use Cases

In this thesis, we present the methodology we followed to systematically derive mutation operators for use case models. More specifically, we first proposed a defect taxonomy defining 104 defect types, based on the IEEE Std. 830-1998 standard. Second, we systematically applied the basic guide words of the standardized Hazard and Operability Study (HAZOP) methodology to define

219 mutation operators. Last, we defined a set of guidelines for devising defect seeding strategies. The proposed methodology was evaluated by two real world case studies and nine case studies from the literature. Results show that all the derived mutation operators for Restricted Use Case Modeling (RUCM) models are feasible to apply and the defect taxonomy is the most comprehensive one to compare with the literature.

4) Search and Similarity Based Secleciton of Use Case Scenarios to Support Requirements Inspection: An Empirical Study

In this theisis, we propose a search-based and similarity-based approach called S^3RUCM, through an empirical study, to select most diverse use case scenarios to enable cost-effective use case inspections. The empirical study was designed to evaluate the performance of four search algorithms (i.e., SSGA, (1+1) EA, AVM, and RS) together with eight similarity functions (i.e., CNT, JAC, GOW, SOK, NLCS, LEV, NW, and SW), through one real world case study and six case studies from the literature. Results show that (1+1) Evolutionary Algorithm together with Needleman-Wunsch similarity function significantly outperformed the other 31 combinations of the search algorithms and similarity functions. The combination managed to select 50% of all the generated RUCM use case scenarios for all the case studies to detect all the seeded defects.

Key words: Real-Time Requirements Modeling; Aspect-Oriented Use Case Modeling; Mutatin Analysis for Use Cases; Use Case Scenarios Selection

目 录

图 目 录

表 目 录

缩 略 语 表

简称	全称	中文解释
AOM	Aspect-Oriented Modeling	面向方面建模
AVM	Alternating Variable Method	交错变量算法
(1+1) EA	(1+1) Evolutionary Algorithm	(1+1)演化算法
CNT	Counting Similarity Function	Counting 相似度函数
CSA	the Combination of Simialrity function and search Algorithm	相似度函数和搜索算法的组合
$CAS_{AVM-(1+1)EA}$	the combination of AVM and (1+1)EA	AVM 和(1+1)EA 组合
$CAS_{AVM-SSGA}$	the combination of AVM and SSGA	AVM 和 SSGA 组合
$CAS_{NW-(1+1)EA}$	the combination of NW and (1+1)EA	NW 和(1+1)EA 组合
$CAS_{SW-(1+1)EA}$	the combination of SW and (1+1)EA	SW 和(1+1)EA 组合
DDR	Defect Detection Rate	缺陷发现率
FV	Fitness Value	适用度函数值
GA	Genetic Algorithm	遗传算法
GOW	Gower-Legendre similarity function	Gower-Legendre 相似度函数
HAZOP	Hazard and Operability Study	危害与可操作性分析
JAC	Jaccard Index similarity function	Jaccard Index 相似度函数
LEV	Levenshtein Distance	Levenshtein 距离
MARTE	Modeling and Analysis of Real-Time and Embedded Systems	实时嵌入式系统的建模与分析
NFP	Non-Functional Properties	非功能属性
NLCS	Normalized Longest Common Subsequence	标准化的最长公共子序列
NMTS	the number of seeded mutants	变种注入的数量
NUCS	the number of use case scenarios	用况场景的数量
NW	Needleman-Wunsch alignment	Needleman-Wunsch 序列排列
OCL	Object Constraint Language	对象约束语言

简称	全称	中文解释
RS	Random Search	随机搜索算法
RT	average Running Time	平均运行时间
RUCM	Restricted Use Case Modeling	限制性用况建模
SIM	Similarity function	相似度函数
SOK	Sokal-Sneath similarity function	Sokal-Sneath 相似度函数
SW	Smith-Waterman alignment	Smith-Waterman 序列排列
SSGA	Steady State Genetic Algorithm	稳态遗传算法
UML	Unified Modeling Language	统一建模语言

中 英 文 概 念 对 照 表

简称	全称	概念定义位置
用况	Use Case	1.1 节，2.1 节
用况规约	Use Case Specification	2.1 节
用况场景	Use Case Scenario	1.2.1.1 节，5.1 节
限制性用况建模	Restricted Use Case Modeling	1.2.1 节
定时图	Timing Diagram	2.2.2 节
面向方面建模	Aspect-Oriented Modeling	1.2.2.1 节，2.2 节
变异分析	Mutation Analysis	4.2 节
变异算子	Mutation Operator	4.2.3 节
变异分数	Mutation Score	4.2.3 节

第1章 绪论

1.1 研究背景

实时系统是一类对时间特性要求十分严格的系统，该类系统的正确性不仅仅依赖于系统计算逻辑结果的正确性，还依赖于产生结果的时间[1]。实时系统广泛应用于航天航空领域、航海领域、海底石油勘探领域、汽车工业领域、健康医疗领域，为我们的日常生活提供各种各样的服务。时间特性需求是实时系统质量的重要衡量标准，实时系统具有严格的时序性、实时性和并发性约束，其时间约束是系统的关键特性，甚至当系统的时间特性无法被满足时，往往伴随灾难性后果带来巨大损失。

在进行综合模块化航电系统(Integrated Modular Avionics (IMA) Systems)这类典型实时系统的开发过程中，工业实践者严格遵循相关的工业标准 DO-178C[2]、DO-332[3]，采用面向对象和模型驱动的技术。这类 IMA 系统通常由一系列计算模块，不同类型的硬件设备，系统与参与者、系统与其他系统或外部系统之间复杂的通讯所构成。在这类 IMA 系统的需求开发过程中，除了复杂的系统功能需求外，还有大量的实时约束、资源限制、通讯协议和异常处理需要被准确捕获并进行精准描述。工业标准 DO-332[3]针对如何使用面向对象技术及其相关技术进行航空系统的开发给出了详细的指导规则。因此，用况建模技术是我们的工业合作伙伴进行需求开发的工业实践技术。

用况[1](Use Case)建模，作为一种对需求进行捕获、描述、分析的建模技术已在工业实践中得到广泛应用[5,6]。用况建模是面向对象开发过程，即“以用况为驱动的，以体系架构为中心的，增量、迭代的开发过程”[4]，的核心。在用况建模的工业实践中，Dr. Tao Yue 提出的 RUCM[7](Restricted Use Case Modeling)用况建模方法表现出显著性的优势：消除自然语言的二义性、模糊性[7]，从用况模型到 UML 分析模型(如 UML 活动图)的自动转换[8]，测试用例的自动生成[9]。RUCM 用况建模方法是一个通用的用况建模框架，它通过提供良好的扩展机制以支持面向不同应用领域的 RUCM 扩展。因此，可以对 RUCM 进行扩展以支持对实时需求的用况建模。

需求是整个系统开发过程中的起始阶段，需要的质量直接影响整个系统的最终质量

[1]术语“Use Case”有两种不同的中文翻译：“用况”、“用例”。本文撰写中采用北京大学邵维忠老师和杨芙清老师提倡的翻译：“用况”。在著作《面向对象的系统分析》[4](第 2 版)中 183 页，两位老师给出了详细的论述。

和维护成本。清晰准确、详实完整的需求是分析、设计、开发和测试等后续软件活动的追溯依据。对于复杂实时系统而言，需求更是繁多复杂，既有各种功能需求又有严苛的时间约束、性能约束等，各种需求相互编织，增加了需求建模人员的工作难度。复杂实时系统(如航空飞行控制系统、通讯控制系统、海底石油勘探系统)对用况建模有特殊的挑战。出于安全和可靠的考虑，这类实时系统在设计上通常都采用双机或多机的设计方案，即系统通常由功能类似甚至一样的多个对等子系统共同构成。在这种系统的用况建模中，系统的某些特性(如通讯媒介、时间约束、资源约束)往往会横穿在不同的用况中，而这些特性往往又是决定系统的关键指标。如果直接对它们进行用况建模，则会造成用况图的混乱和用况规约片段的冗余，导致用况模型很难被理解和可复用。因此，需要使用一种建模技术能够将这些横切关注点(如实时约束)进行分离，对其进行独立建模和管理。

面向方面建模(Aspect-oriented Modeling (AOM))是在面向方面编程(Aspect-oriented Programming (AOP))的概念基础上对软件开发在设计分析层次上的延伸，通过面向方面的需求建模可以实现在需求阶段将时间特性等非功能需求与功能需求的分离，通过 AOM 能将实时系统最重要的时间特性作为一个独立的方面来进行需求建模，为后续针对时间特性的分析和设计提供直接支持，同时通过建立的时间需求模型可以进行时间方面的单独管理。另外，软件需求规约也是需求过程中比较重要的一个环节。清晰、完整的需求描述是后续开发活动和测试活动的重要依据和溯源。对需求规约方法的选择通常是在可读性和推理能力之间的一种博弈[10]，自然语言非常灵活便于沟通但是不能进行推理；形式化的表示方法能够捕获精确的语义，支持丰富的验证技术，但是对用户要求较高不太好用，在工业实践中很少在需求开发阶段被直接使用；半形式化的模型(如 UML 用况模型)对于不同利益相关者之间的交流非常有用同时又支持一定的仿真、模拟能力，是工业实践的首选。因此，可以通过对使用面向方面的建模方法对 RUCM 用况建模进行扩展，使其在用况建模过程中支持横切关注点的分离，从而实现对复杂实时系统的用况建模。

需求评审，即软件评审[11]技术针对需求文档的一种特殊应用，是已被实证研究证实了并被广泛接受的一种有效技术，它可以较早的对需求中的缺陷进行识别和检测[12]，是一种重要的需求验证技术。研究[13]中指出，当评审过程被正确执行后，95%的缺陷可以

在测试阶段之前被发现。另外一个成功的案例 JPL(Jet Propulsion Laboratory)喷气推进实验室，通过采用 300 次评审所发现的缺陷为项目节省了 750 万美金[14]。自从文献[15]对软件评审进行首次阐述，研究学者提出了各种不同的需求评审技术。基于检查列表的阅读技术[12](Checklist-based reading (CBR)[12])，通过设计一系列问题来指导需求评审员执行评审过程。基于缺陷的阅读技术[16] (Defect-based reading (DBR)[16])，针对某类特定的需求缺陷设计相应的检查列表以辅助评审人员执行评审过程。基于视角的阅读技术(Perspective-based reading (PBR)[17])，从不同的利益相关者的角度(如开发者、测试者的角度)进行相应的评审活动。基于使用的阅读技术[18] (Usage-based reading[18])，对用况进行优先化并从系统终端用户的视角进行需求评审。

需求评审是一个高度依赖于评审人员的过程，个人的经验、熟练程度都直接影响具体评审方法的有效性。针对不同的评审方法的有效性的对比分析，研究者经常给出不一致的结论[19]。另外，在针对不同评审技术的有效性分析的实证研究中，研究学者都采用故障注入的方式预先创建相应的需求缺陷[19]。然而目前的实证研究实验中即缺少一种系统化而精确的需求缺陷分类，也没有给出一种系统化进行故障注入的方法，从而导致相应的实验几乎不能重现。因此，针对不同评审方法的对比分析方法中的结论不一致性和需求缺陷注入的问题，需要一种系统化进行需求缺陷创建的方法。

需求评审又是一个时间成本比较高的过程。例如，文献研究[14]中，JPL 的评审中每一次评审的平均花费是 28 小时，按此计算 JPL 评审的时间总代价是 8400 小时也就是一年。在评审资源(工程时间、人力、项目预算)有限的约束下，大量的需求场景迫使需求分析人员必须采取有效的选择策略进行高效的需求评审。对于复杂实时系统而言，需求更是繁多复杂，既有各种功能需求又有严苛的时间约束、性能约束等，各种需求相互编织，加大了需求评审的难度。因此在大量需求场景与有限开发成本的矛盾下，研究用况场景的择优选择策略对于高效实施需求评审降低系统开发成本有着重大支撑作用。

综上所述，研究面向方面的实时性需求建模是针对实时系统在需求开发阶段的一项重要内容。在软件需求建模阶段实现关注点的分离，能够帮助需求人员将时间特性从功能需求中分离出来从而专注时间特性的需求建模，并建立一个时间需求模型可以进行单独管理，提高需求模型的可复用。从需求评审的支撑角度出发，研究系统化创建用况需求缺陷的方法，支撑对不同的用况评审技术(如基于检查列表的技术、基于视角的阅读技

术)进行客观公正、有效可靠的评估，同时又可以支持评审实验过程的可再现。针对评审资源(时间-成本)受限的需求评审，研究基于启发式搜索的用况场景择优选取，为需求评审的高效开展提供前期支持。在需求阶段将需求模型自动转换成基于 MARTE 的时间特性分析模型可以支持系统时间分析在需求阶段的开展，为后续的分析、设计提供支持，从而降低系统开发和维护的成本。

1.2 相关研究

本节从解决本文研究问题的角度对相关的研究工作进行调研。首先，从实时系统用况建模的角度，对 RUCM(Restricted Use Case Modeling)用况建模方法进行研究；其次，从实时特性独立建模的角度，对支持关注点分离思想的面向方面建模方法的相应研究工作进行综述；再次，从需求评审支撑的角度(即如何系统化的创建需求缺陷)，调研变异分析技术在模型驱动工程的应用；最后，从需求评审支撑的角度(即如何选择部分需求场景作为评审源)，充分调研基于搜索的软件工程(SBSE)中的需求选择的相关研究。

1.2.1 RUCM(Restricted Use Case Modeling)用况建模方法

本小节对 RUCM 用况建模方法进行详细阐述，1.2.1.1 节描述了 RUCM 用况规约模板，1.2.1.2 节陈述了 RUCM 的限制规则，1.2.1.3 节阐述了 RUCM 的形式化机制 UCMeta，阐述了 RUCM 的实验研究。

1.2.1.1 RUCM 用况规约模板

用况(Use Case)建模，作为一种对需求进行捕获、描述、分析的建模技术已在工业实践中得到广泛应用[5,6]。在工业实践中通常采用一个模板对用况规约进行组织和描述，不同的用况模板(如[20-22])被设计来解决这个问题。这些用况模板都有一些通用的内容：用况名字、用况基本描述、前置条件、后置条件、基本流、备选流。Dr. Tao Yue[23,8,24]基于当前用况建模的学术研究和工业实践，提出了 RUCM 用况建模方法。除了标准的用况图，RUCM 提供一个结构化的用况规约模板进行用况描述，同时针对自然语言的使用设计了 26 条限制规则用以最大程度的减少自然语言引入的模糊性和二义性。此外 RUCM 还采用 UCMeta 元模型建模的形式化机制，支持模型分析和转换。

表 1 展示了 RUCM 用况规约模板，从中可发现它是一个结构化的描述模板，共有 11

个组成部分。前七个组成要素分别是：用况名字、用况基本描述、用况前置条件、首要参与者、辅助参与者、与其他用况的依赖关系即«Include»或«Extend»关联的用况、与其他用况的泛化关系。这七个用况描述项是用况描述的基本信息，同时也捕获了用况图中对应用况的建模信息。

表 1 RUCM 用况规约模板*

Use Case Name	The name of the use case. It usually starts with a verb.	
Brief Description	Summarizes the use case in a short paragraph.	
Precondition	What should be true before the use case is executed.	
Primary Actor	The actor which initiates the use case.	
Secondary Actors	Other actors the system relies on to accomplish the services of the use case.	
Dependency	Include and extend relationships to other use cases.	
Generalization	Generalization relationships to other use cases.	
Basic Flow	Specifies the main successful path, also called “happy path”.	
	Steps (numbered)	Flow of events.
	Postcondition	What should be true after the basic flow executes.
Specific Alternative Flows	Applies to one specific step of the basic flow.	
	RFS	A reference flow step number where flow branches from.
	Steps (numbered)	Flow of events.
	Postcondition	What should be true after the alternative flow executes.
Global Alternative Flows	Applies to all the steps of the basic flow.	
	Steps (numbered)	Flow of events.
	Postcondition	What should be true after the alternative flow executes.
Bounded Alternative Flows	Applies to more than one step of the basic flow, but not all of them.	
	RFS	A list of reference flow steps where flow branches from.
	Steps (numbered)	Flow of events.
	Postcondition	What should be true after the alternative flow executes.

*该用况模板引用自文献[24]

RUCM 事件流是对不同用况场景的结构化组织单元。在 RUCM 中事件流有两大类：基本流(Basic Flow)和备选流(Alternative Flows，其中备选流(Alternative Flow)又细化为：特定流(Specific Alternative Flows)、组界流(Bounded Alternative Flows)、全局流(Global Alternative Flows)。每一个事件流都有一个后置条件(Postcondition)，描述当前事件流结

束后系统的状态或行为。

基本流(Basic Flow)：描述了用况的一个主要的成功的执行路径，它通常不包含任何分支[25]，而在备选流中描述相应的分支和条件。一个基本流是由一组顺序执行的动作语句构成，在 RUCM 中一个用况有且只有一个基本流。动作语句实际上就是系统与参与者之间的交互，而每个动作语句所描述的交互只能是以下五种之一：1)首要参与者→系统：首要参与者向系统发起一个请求或者发送一个数据给系统。2)系统→系统：系统对接收的请求或数据进行验证。3)系统→系统：系统改变内部状态，例如记录或修改数据等。4)系统→首要参与者：系统向首要参与者做出反应或返回处理结果。5)系统→辅助参与者：系统向辅助参与者发送请求。其中前四种交互方式是从[26]借鉴而来的。

所有的动作语句都是顺序标号的，也就是说一个动作语句执行时它前面所有的动作语句已经执行过。如果需要表示条件、迭代、并发等情形，RUCM 通过预定义的关键字机制进行处理，这些关键字是一些特殊的 RUCM 语句，并在限制规则中进行特殊说明。

备选流(Alternative Flows)描述了用况执行过程中的其他场景或分支，它们可能会成功执行也可能是用况执行失败的场景。任何一个备选流是从某个流(基本流或备选流)中在一定条件下产生的分支，因此每个备选流都会有一个 RFS(1.2.1.2 节)关键字标识其相应的分支点，其中分支条件由限制规则(R20 和 R22，1.2.1.2 节)进行详细说明。RFS 中指定的语句在 RUCM 中叫做“条件”语句。在 RUCM 用况规约中的描述语句只有“条件”语句和“动作”语句两类。

与基本流一样，备选流也是由一组顺序执行的语句构成，每个语句也都进行顺序标号。备选流又细化为：特定流(Specific Alternative Flows)、组界流(Bounded Alternative Flows)、全局流(Global Alternative Flows)，这种分类方式是从文献[27]中引入的。一个特定流(Specific Alternative Flows)是从一个流(基本流或备选流)中的一个特定的“条件”语句中分支出来的。一个组界流(Bounded Alternative Flows)是对多个来自同一个流或不同的流中的“条件”语句进行统一处理的逻辑描述。一个全局流(Global Alternative Flows)对应着基本流中任何一个语句，它用于描述一些系统全局性的操作或反馈，例如参与者突然中止当前的业务操作。

1.2.1.2 RUCM 限制规则

RUCM限制规则分为两类：一类用于限制自然语言的书写使用(表2-表3，引用自文献

[24])，另一类借助RUCM关键字用于描述控制信息(表4，引用自文献[24])。

R1-R7(表2)被设计来减少自然语言中的模糊性和二义性，这些规则仅适用于所有的“动作”语句，而不用于“条件”语句、前置条件、后置条件。例如，R1中规定，在用况规约中的每个“动作”语句中的主语只能是“当前系统”或者一个“参与者”。R2中规定必须要顺序描述每一个事件流，即要求需求人员必须要理清系统处理的逻辑流程。用况规约中不允许存在参与者与参与者之间的交互(R3)。

表 2 RUCM 限制规则 R1-R7

#	Description	Explanation
R1	The subject of a sentence in basic and alternative flows should be the ystem or an actor.	Enforce describing flows of events correctly. These rules conform to our use case template (the five interactions).
R2	Describe the flow of events sequentially.	
R3	Actor-to-actor interactions are not allowed.	
R4	Describe one action per sentence. (Avoid compound predicates.)	Otherwise it is hard to decide the sequence of multiple actions in a sentence.
R5	Use present tense only.	Enforce describing what the system does, rather than what it will do or what it has done.
R6	Use active voice rather than passive voice.	Enforce explicitly showing the subject and/or object(s) of a sentence.
R7	Clearly describe the interaction between the system and actors without omitting its sender and receiver.	

R8-R16(表3)可以使用于所有的“动作”语句、“条件”语句、前置条件、后置条件以及用况基本描述信息中的所有描述语句。R8-R10和R16 用于减少用况规约中的模糊性和二义性。R11-R15不仅可以减少用况规约的模糊性，同时也方便进行模型转换(用况模转换成其他的UML分析模型(如UML活动图))。这两组规则(表2-表3)是被研究学者广泛推荐的书写用况规约的有效规则[27,20,28]，例如R9要求需求人员使用一致的术语和词汇进行用况规约的书写。

表 3 RUCM 限制规则 R8-R16

#	Description	Explanation
R8	Use declarative sentence only. “Is the system idle?” is a non-declarative sentence.	Commonly required for writing UCSs.
R9	Use words in a consistent way.	Keep one term to describe one thing.
R10	Don’t use modal verbs (e.g., *might*)	Modal verbs and adverbs usually indicate uncertainty; Instead, metrics should be used if possible.
R11	Avoid adverbs (e.g., *very*).	
R12	Use simple sentences only. A simple sentence must contain only one subject and one predicate.	Facilitate automated natural language parsing and reduce ambiguity.
R13	Don’t use negative adverb and adjective (e.g., *hardly*, *never*), but it is allowed to use *not* or *no*.	
R14	Don’t use pronouns (e.g. *he*, *this*)	
R15	Don’t use participle phrases as adverbial modifier. For example, the italic-font part of the sentence “ATM is idle, *displaying a Welcome message*”, is a participle phrase.	
R16	Use “the system” to refer to the system under design consistently.	Keep one term to describe the system; therefore reduce ambiguity.

表 4 RUCM 限制规则 R17-R26

#	Explanation	#	Explanation
R17	INCLUDE USE CASE	R22	VALIDATE THAT
R18	EXTENDED BY USE CASE	R23	DO-UNTIL
R19	RFS	R24	ABORT
R20	IF-THEN-ELSE-ELSEIFENDIF	R25	RESUME STEP
R21	MEANWHILE	R26	Each basic flow and alternative flow should have its own postconditions.

除 R26，其他剩余的限制规则(表 4)都是设计来对用况中的控制信息进行描述的。R17 和 R18 使用关键字对用况的依赖关系(«Include»或«Extend»)进行描述。R19 描述关键字 RFS，对备选流的分支点(一个或多个“条件”语句)进行精确描述。规则 R20-R23 对不同的控制信息进行描述：逻辑条件语句(IFTHEN-ELSE-ELSEIF-ENDIF)，并发动作语句(MEANWHILE)，条件检查语句(VALIDATES THAT)，迭代控制语句(DO-UNTIL)。其中 IF-THEN-ELSE-ELSEIF-ENDIF 又可以细化为三种不同的应用场景：1) IF-THEN-ENDIF(只可以在同一个流中使用)，2) IF-THEN-ELSE-ENDIF(整个结构在同一个流中；

IF-THEN 结构在一个流中，ELSE 在相应的备选流中)，3) IF-THEN-ELSEIF-THEN-ELSE-ENDIF(整个结构在同一个流中； IF-THEN 结构在一个流中，ELSEIF-THEN-ELSE-ENDIF 在相应的备选流中)。关键字 VALIDATE THAT (R22)说明相应的条件必须要经过系统验证且成立时才会执行下一步的“动作”语句。R24 和 R25 分别描述关键字 ABORT 和 RESUME STEP，而且这两关键字只能够使用在备选流中。其中关键字 ABORT(R24)说明用况在当前的备选流中异常结束，而关键字 RESUME STEP (R25)指明当前的备选流需要返回到相应的分支流中的哪个语句。

1.2.1.3 UCMeta 形式化机制

UCMeta 是 RUCM 采用的形式化机制，它采用基于 MOF[29](Model Object Facility)的元模型建模。UCMeta 是连接文本用况描述和 UML 分析模型的中间模型，也是从 RUCM 用况模型到 UML 分析模型(如类图、活动图)转换的依据。图 1(引用自文献[24])展示了 UCMeta 的结构，从中可以发现 UCMeta 是三层架构：顶层是 UCMeta 包，它使用了中间层的包：UML::UseCases，UCSTemplate，SentencePatterns，SentenceSemantics。包 UCSTemplate 又使用了底层的包：SentenceStructure。

图 2(引用自文献[24])是 UCSTemplate 元模型，它对 RUCM 用况规约模板(1.2.1.1 节)中的相关概念进行建模。如图 2 所示，这些元模型与表 1 中的元素是相对应的。UML 包 UML::UseCases 中的元类类型 UseCase 与元类类型 UseCaseSpecification 之间存在关联(association)关系，即每个 RUCM 用况都有一个相应的用况规约。UseCaseSpecification 包含一个基本描述(BriefDescription)，一个前置条件(Preconditon)，一个或多个事件流(FlowOfEvents)，一个首要参与者(primaryActor)，零到多个辅助参与者(secondaryActors)。事件流又分为两类：基本流(BasicFlow)和备选流(AlternativeFlow)。每个用况有且仅有一个基本流，可以有零或多个备选流。备选流(Alternative Flow)又细化为：特定流(Specific Alternative Flows)、组界流(Bounded Alternative Flows)、全局流(Global Alternative Flows)。

SentencePatterns 包(图 1)定义了 RUCM 语句的 8 种不同类型，这些分类是依据语义学(如[30])和语法规则(如[31])进行定义的。

1) SV (主语-谓语)模式。

2) SVC (主语-谓语-补语)模式。

3) SVCC (主语-谓语-补语-补语)模式。

4) SVDO (主语-谓语-直接宾语)模式。

5) SVDOC (主语-谓语-直接宾语-补语)模式。

6) SVIODO (主语-谓语-间接宾语-直接宾语)模式。

7) SVIODOC (主语-谓语-间接宾语-直接宾语-补语)模式。

8) SLVSubjectCompltl (主语-系动词-主语补语)模式。

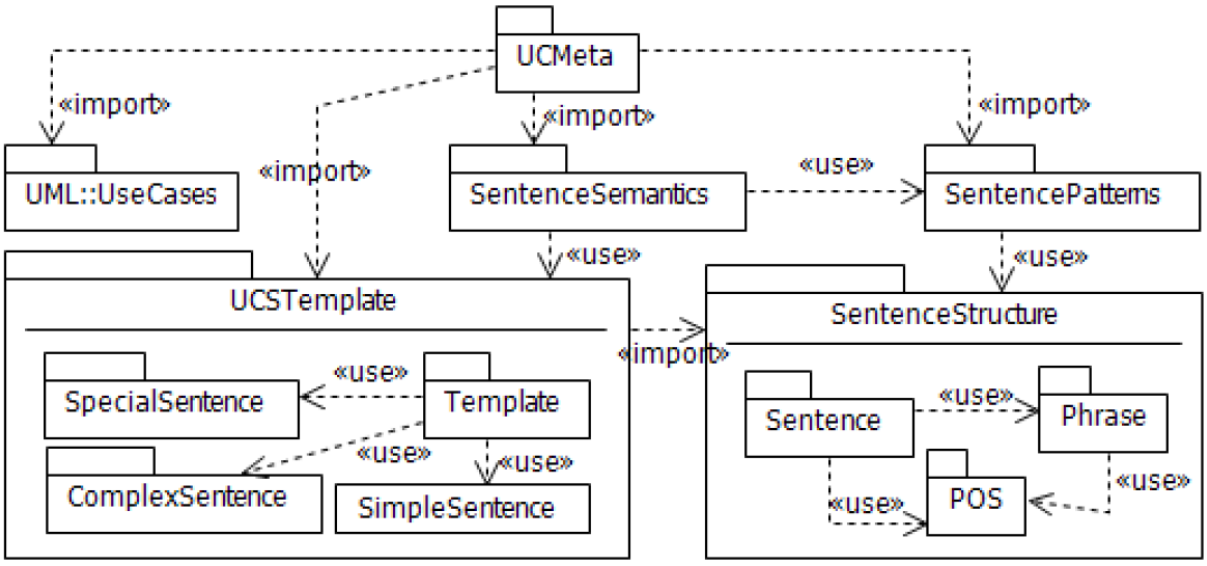

图 1 UCMeta 的结构图

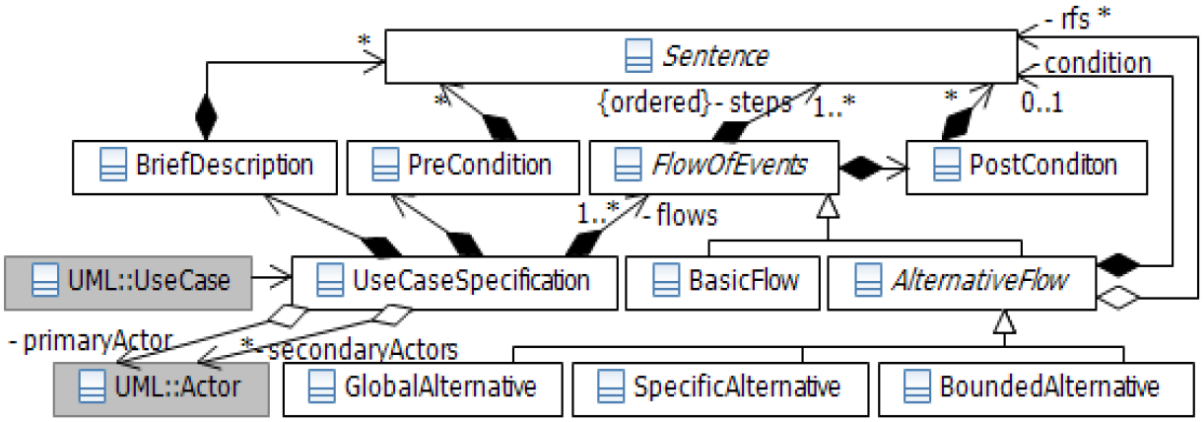

图 2 用况规约模板的 UCMeta

1.2.1.4 RUCM 的实证研究(Empirical Study)

为了对 RUCM 方法的有效性进行全面评估，Dr. Tao Yue 采用实证研究(Empirical Study)的方法对 RUCM 设计了一系列的受控实验。文献[23]中进行了两个受控实验对 RUCM 的可用性、用况规约模板、限制规则进行了相应的评估，实验结果显示 RUCM 非常易于使用并能显著提高用况规约的质量。文献[169]针对从 RUCM 用况模型到 UML

分析模型的模型转换进行实验，结果显示 RUCM 生成的 UML 分析模型的质量比传统方法生成的 UML 分析模型的质量有显著性的提高。

此外，RUCM 用况建模方法已被我们的工业合作伙伴应用到航空领域、海底石油勘探领域的工业实践中并进行初步的研究，但是标准的 RUCM 用况建模方法还不能支持面向实时领域的特性建模，也不支持关注点分离的需求建模，本文的研究将会从这两个方面对标准 RUCM 建模方法进行扩展，从而解决工业实践中的现实问题。

1.2.2 面向方面建模

本节从面向方面建模的角度对相关的研究进行调研，1.2.2.1 节陈述了面向方面需求工程中的相关研究，1.2.2.2 节调研了面向方面实时特性建模的研究工作。

1.2.2.1 面向方面需求工程

面向方面需求工程((Aspect-oriented Requirements Engineering(AORE))致力于需求工程中的横切关注点(crosscutting concerns)的识别和分析，从而确定横切关注点的潜在影响。在 AORE 中将横切关注点定义为“方面”(Aspect)或“切面”。AORE 方法通常是对传统的需求工程技术进行扩展，以支持对横切关注点的识别，模块化，组合和分析，而这种支持是现代多数需求工程技术所缺失的。虽然某些技术，比如基于目标的需求方法[32]，支持对非功能需求的模块化分析，但缺乏有效的组合机制来捕获和描述系统非功能需求之间以及非功能需求和功能需求之间复杂的依赖和交互[33]。面向方面的需求方法，则较为系统地提供了一种需求横切关注点模块化、编织、分析的手段。

Grundy[34]提出了一种面向方面的需求方法(AOCRE)对基于构件的系统进行需求建模。该方法旨在识别和描述系统中每个构件提供或需要的功能需求和关键的非功能需求。它从识别构件的“方面”开始，针对每个构件，识别在哪些“方面”该构件为其他构件提供服务，或者需要其他构件提供的服务。该方法提升了系统的可重用性以及可扩展性，基于这些构件构建的系统大大提高了数据和行为的职责分配。然而，该方法是特定于基于构件的开发方法，并没有在通用软件开发方法中得到应用。

Whittle[35]等提出了一种基于场景需求的面向方面建模方法。该方法支持独立描述“方面”场景和基场景，并将两种场景进行编织。“方面”场景，即横切在基场景中的场景，采用交互模式规约(Interaction Pattern Specifications: IPSs)进行建模。基场景采用 UML 顺

序图进行建模。每个“方面”场景以及对应的基场景都被转化为一系列状态机，随后基于状态机进行编织。通过编织“方面”状态机以及基状态机，该方法可以辅助需求工程人员从整体上捕获系统需求。该方面很好的支持了模块化以及追踪性，但是缺乏可扩展性。开发人员必须为每个“方面”以及该“方面”横切的场景提供绑定语句，而且该方法也不支持“方面”场景的识别。

Brito[36]等人提出了一种“方面”需求的集成方法，来处理关注点分离、模块化、建模表示、模型编织。该方法使用 UML 用况图、交互图和类图对关注点进行建模和描述。文献[37]提出一种面向方面的需求建模方法，期望在软件开发的早期阶段应用面向方面的开发模式。文献[38]提出了一种面向方面的方法来支持 UML 活动图中横切关注点的建模。

综上所述，已有的研究工作对面向方面的建模方法进行系统的研究，研究者借助不同的 UML 模型(如类图、状态机图)实现横切关注的分离和建模。本文的研究中需要把面向方面的建模方法应用到 UML 用况建模中，即实现支持面向方面建模能力的用况建模。因此，如何定义“方面”参与者(Actor)、“方面”用况(Use Case)、“方面”用况规约(Use Case Specification)，并实现它们与基用况模型进行编织是本文研究面向方面建模的主要内容。上述已有的研究工作为本文的研究提供相应的支持：如何在“方面”模型中描述与基模型相关联的模型元素(“切入点”)；如何定义在基模型中引入的新的模型元素(“通知”)；如何实现“方面”模型与基模型的编织融合。

1.2.2.2 面向方面的实时特性建模

Wehrmeister[39]针对嵌入式实时系统提出了一种建模框架 DERAF，希望在设计的早期阶段通过分离关注点的方式捕获非功能需求，并采用 RT-UML[40]对时间特性进行建模。DERAF 是一种非功能需求的导出方法，不具备需求规约的能力，只能辅助建模者识别非功能需求。

文献[41]采用面向方面的建模方法对分布式实时系统中的时间特性进行建模。通过扩展 UML 类图和 UML 顺序图实现对系统时间特性的“方面”建模。然而该方法不支持对需求规约的能力，只能看做是一种识别实时性非功能需求的方法。随后在文献[42]中，他们又将时间“方面”细化为多个子“方面”，即“模糊”的时间子“方面”、 “不确定”的时间子“方面”、 “确定”的时间子“方面”，并采用随机实时时序逻辑(SQTL)和模糊时间 Petri 网

进行形式化定义，最后借助时间自动机实现“方面”编织。该方法是从系统架构层面实现时间特性与系统行为的独立建模，而不是在需求阶段从功能需求与非功能需求的视角实现需求的独立建模。

上述研究针对系统时间特性采用面向方面的建模方法从不同的视角展开了研究，文献[39]提供了一种面向方面的需求导出(Elicitation)方法，但是不能提供有效的需求规约(Specificaiton)，这对于复杂实时系统的需求开发还是远远不够的。文献[41]和文献[42]从系统体系结构设计的视角，实现了系统静态结构模型(UML 类图)和系统行为模型(UML 顺序图)分别与时间特性的独立建模，并对时间约束进行形式化的定义，但是未针对需求建模进行面向方面的建模研究。本文的研究内容是在需求建模即用况建模中实现面向方面建模的思想，将实时特性建模为系统的非功能需求并为其设计相应的“方面”模型，最后将这些实时特性的“方面”模型与基用况模型进行编织。

1.2.3 变异分析在模型驱动工程中的应用

变异分析又叫做变异测试，最初是被设计来对测试用例的有效性进行评估的一种技术[43]。变异分析被广泛地应用于各种编程语言的源程序中(即程序变异[44])，如 C[45]、Java[46]、Ada[47]。变异分析也被应用到程序规约说明中，即规约变异(*Specification Mutation*)[44]，例如，文献[48]中作者使用变异分析技术对有限状态机进行验证。文献[44]是针对变异分析的最新文献综述，文献[44]的研究显示研究学者对于程序变异的研究要比对规约说明变异的研究多。变异分析已应用到模型驱动工程(Model-Driven Engineering)中[49,50]，特别是模型驱动测试和模型转换两方面的研究。Schlick[51]等提出一种基于缺陷和 UML 模型的测试用例生成方法。该方法针对 UML 状态机模型设计了四类变异算子，从而实现测试用例的自动生成(原文中把这些变异算子通称为故障模型[51])。替换触发事件(*Replacing trigger event*)，改变状态机当前迁移的事件源；设置状态迁移条件“永真”(*Setting transition guard to TRUE*)，这会导致当前触发事件对应的状态迁移总是会执行；设置状态迁移条件“永假”(*Setting transition guard to FALSE*)，这会导致当前触发事件对应的状态迁移永远不被执行；改变迁移的目标状态(*Aiming transition at another state*)，这会导致触发事件将系统迁移到另外的状态。

文献[52]中，作者针对数据处理系统使用基于模型的变异分析，设计一种自动生成复杂错误的测试数据的方法。该方法针对 UML 类图，定义了相应的变异算子：重复类的

对象实例(*Class Instance Duplication*)给定一个类，重复它的对象实例。 移除类的对象实例(*Class Instance Removal*)给定一个类，删除它的一个对象实例。交换类的对象实例(*Class Instances Swapping*)，给定一个类，将它的两个对象实例的位置进行互换。随机替换属性值(*Attribute Replacement with Random*)，给定一个类的实例对象的属性标识符，随机选择一个相应的属性值替换原来的属性值。属性值的边界值替换(*Attribute Replacement using Boundary Condition*)，给定一个类的实例对象的属性标识符，使用属性的可能边界取值±1(如，最小值-1、最大值+1)替换原来的属性值。

依据模型转换中可能出现的故障，文献[53]提出了一系列模型转换的变异算子。导航(*Navigation*)相关的变异算子(即影响输入模型和输出模型之间关联关系的操作)：同一个“类”的修改引起的关系改变；不同“类”的修改引起的关系改变；“删除”导致的关系改变；“添加”导致的关系改变。模型筛选相关的变异算子(即给定一个导航，选择相应的合适模型元素)：“波动”导致的模型筛选改变；“删除”导致的模型筛选改变；“添加”导致的模型筛选改变。 模型创建相关的变异算子(即创建一个输出模型相关的变异算子)：“类”兼容性的替换；“删除”导致“类”的关联发生改变；“添加”导致“类”的关联发生改变。

文献[54]论述道：变异分析可以应用到不同的软件制品(如，程序语言、代数规约、UML 模型转)，基于它们各自的句法描述(例如，程序语言通常使用 EBNF[55]进行定义)。作者将传统的变异测试定义为一种通用的基于语法的变异分析，即给定一种软件制品(程序语言、UML 模型)，基于其应的句法定义便可以设计相应的变异算子。

综上所述，变异分析是一种评估测试用例有效性的重要技术，程序变异[44](*Program Mutation*)和规约变异[44](*Specification Mutation*)得到研究学者的广泛认可。研究学者已将变异分析应用到 UML 类图、UML 状态机模型用以创建测试用例。这些研究工作的背后思想可以借鉴到需求评审的支撑中，通过创建变异算子提供一种系统化创建需求缺陷的方法，从而促进对不同的需求评审技术客观公正的评估。

1.2.4 基于搜索的需求选择

Harman[56]等对基于搜索的软件工程的研究进行了全面系统的文献综述，并指出不同的搜索算法被应用于解决软件工程中的各种问题，例如需求选择和优化问题[57]、需求分配问题[58]、需求优先化的问题[59]。NRP问题(The Next Release Problem (NRP)[57])是一个典型的使用搜索技术解决的需求选择问题，即通过选择一部分需求实现资源约束和用户

需求之间的平衡。文献[57]使用了五个不同规模的问题和三类不同的搜索策略(精确技术、贪心算法、局部搜索(爬山算法、模拟退火算法))对NRP进行阐述，其中最小规模的问题有100个客户和140个任务，最大规模的问题有500个客户和3250个任务。实验结果显示：精确技术对于解决最小规模的问题性能显著好，对于规模较大的问题，模拟退火算法的性能显著的好。Baker[59]等采用贪心算法和模拟退火算法解决软件构件选择的NRP问题，实验结果显示这两个算法的性能要比领域专家的判断显著得好。

NRP问题又被发展为多目标的NRP(Multi-Objective Next Release Problem (MONRP)[60])。MONRP主要被设计来挑选一些客户需求，以使得公司利益最大化同时把实现这些客户需求的成本最小化。Zhang[60]采用四种不同的搜索算法研究MONRP：随机搜索、单目标遗传算法、帕累托遗传算法、非优势排列遗传算法。实验结果显示：非优势排列遗传算法比较适合于解决MONRP问题。Saliu和Ruhe[61]提出一种决策支持方法将NRP问题定义为一种双目标优化问题，并采用ε-约束算法[62]对双目标优化问题进行求解。在文献[63]中，作者采用演化算法和量子计算研究MONRP问题。 文献[64]中，作者采用蚁群算法(ACO)解决NRP，他们将ACO与贪婪随机适应搜索策略(GRASP)、非优势排列遗传算法(NSGA)进行对比。实验结果显示：GRASP的性能最差， NSGA难以使用，ACO可以有效的解决NRP。

Greer和Ruhe[65]提出一种软件发布计划(RP)问题。RP旨在选择和分配一些新的特性并对需求进行修改从而支持一系列连续的产品发布计划。RP主要研究两方面的内容：哪些需要发布(what)和什么时候发布(when)[61]。Li[66]等提出一种集成需求选择和分配的方法解决RP问题，他们采用01背包模型[67]对需求选择进行解决。文献[68]中，作者将基于主题的RP问题形式化为一个双目标优化问题，并采用多目标优化算法NSGA-II[69]进行解决。

Li[70]等针对 CPS 系统(Cyber-Physical Systems)提出一种需求分配的方法，该方法在将需求分配给不同利益相关者时，实现分配者对需求熟悉程度的最大化同时平衡相应的工作量。她们分别设计单目标和多目标的优化策略，并选用一个工业案例和 120 个人工问题进行实验。实验结果显示：对于单目标优化问题的解决，(1+1) EA 的性能最好， 而在多目标优化的问题解决中 NSGA-II 的性能最好。

综上所述，基于搜索的软件工程(SBSE)将不同的工程实践问题(如，需求选择、需求

分配)形式化为一个搜索优化问题，通过设计相应的适应度函数并采用不同的搜索算法(如(1+1) EA，NSGA-II)进行优化方案的求解。本文从需求评审支撑的角度出发，在评审资料(可投入的人力和工程时间)受限的条件下，优先选择一部分需求作为评审源并期望这些选择的需求能够涵盖尽可能多的需求缺陷，从而支持需求评审的开展。因此，该问题的解决可以借鉴基于搜索的软件工程(SBSE)的方法，采用搜索算法进行择优选择。

1.3 研究目标及关键问题

本文从工业实践中的问题出发，分别从需求建模和需求评审支撑的视角开展相应的研究工作。以支持实时系统的用况建模，继而实现实时特性需求的独立建模；支持系统化的进行用况需求缺陷的注入；支持有效选择用况场景以实现成本-效益的需求评审的开展；为研究目标并通过工业案例进行有效性评估。具体分为四个子目标：1) 辅助需求人员在系统需求开发阶段有效地使用用况建模技术识别实时特性需求并对实时系统进行需求建模和需求规约；2) 辅助需求人员在面对复杂实时系统时，支持实时特性需求的关注点分离，从而提高用况模型的可重用、可维护，并提高需求开发的效率和质量；3) 辅助需求评审人员系统化的为用况模型创建各种需求缺陷，从而为评估不同的评审方法提供支持；4) 辅助需求评审人员自动获取有效的用况场景，在评审资源(人力、时间)受限时为成本-效益的需求评审提供评审源。

为了实现上述研究目标，本文需要解决以下四个关键问题：

(1) 如何扩展 RUCM 用况建模方法，以实现对实时系统的需求建模？

为实现 RUCM 对实时系统的用况建模，需要研究实时系统的需求特性，识别出那些需要在需求阶段就需要捕获、描述的系统特性和约束。具体到用况建模中，使用元模型建模的技术解决以下的问题：扩展参与者元模型，从而辅助需求人员识别实时系统的不同参与者(例如，外部设备、系统资源)；扩展用况元模型，辅助需求人员识别不同的实时任务(例如，周期性任务、不规则实时性任务)；引入新的约束元模型(例如，资源约束、时间约束)，辅助需求人员对这些实时特性进行准备捕获和精确规约。

(2) 如何将面向方面的建模思想应用到目标(1)实现的用况建模中，从而支持实时特性需求与基用况模型的分离？

为实现实时特性需求与基用况模型的分离，定制面向实时特性的用况“方面”建模语

言是关键。为了将基用况模型的相关元素与“方面”用况模型进行关联，该“方面”建模语言需要为需求人员提供一种描述具有共性特征的一类用况建模元素(如用况、参与者、实时约束)的方法；该建模语言还需要定义用况“方面”模型，即描述“方面”建模元素以及“方面”用况模型如何与基用况模型进行编织，以支持实时特性需求在基用况模型中的实例化。此外，针对面向方面的用况建模，需要给出相应的建模规则用以辅助需求人员进行建模实践。

(3) 如何系统化的为用况模型注入需求缺陷，辅助需求评审人员对不同的需求评审方法进行客观公正的评估？

为实现对不同的需求评审方法进行客观公正的评估，对其进行系统化的故障注入是相关实验设计中的首要环节。为了支持系统化的进行缺陷故障注入，首先需要给出全面、明确的缺陷分类定义，这些不同类型的缺陷实例是评估不同评审方法的直接指标；然后需要针对用况模型分析相应的构成元素，以确定如何通过对用况建模元素进行修改实现不同需求缺陷实例的创建；最后需要研究故障生成策略，为整个需求缺陷注入的过程提供相应的指导规则，采用一种成本-效益的方式进行需求缺陷的注入。

(4) 如何自动选择有效的用况场景，辅助需求评审人员在有限的评审资源(人员、时间)下进行成本-效益的用况评审？

需求评审是一个以人为主导的活动过程，在评审资源(人员、时间)受限的条件下，如何选择相应的需求项进行开展是一个工业实践问题。具体到复杂实时系统的需求开发中，一个系统往往会有大量的用况场景，这些用况场景描述了系统在不同条件下面对不同的系统输入进行的一系列动作响应和系统反馈。如何从大量的用况场景中挑选出一部分用况场景并包含尽可能多的需求缺陷，从而支持用况评审的有效开展是一个关键挑战。

1.4 研究框架及内容

用况建模及用况评审是实时系统需求开发阶段的重要内容。首先研究如何基于RUCM 用况建模方法实现对实时系统的用况建模，从而可以辅助需求人员创建高质量的用况规约；面对复杂实时系统，采用关注点分离技术实现实时性需求的独立建模，继而支持面向方面的实时性需求的用况建模；依据工业标准(IEEE 830-1998 Standard)系统化的定义用况缺陷分类，依据工业标准(IEC 61882) 创建相应的用况变异算子，从而实现

系统化的缺陷注入的方法，为客观公正的评估不同的需求评审方法提供支持；在评审资源(人力、时间)受限下，基于相似度和搜索算法选择有效的用况场景为成本-效益的需求评审提供评审源。

图 3 展示了本文的研究框架，本文的研究内容具体细化为以下四个子项：

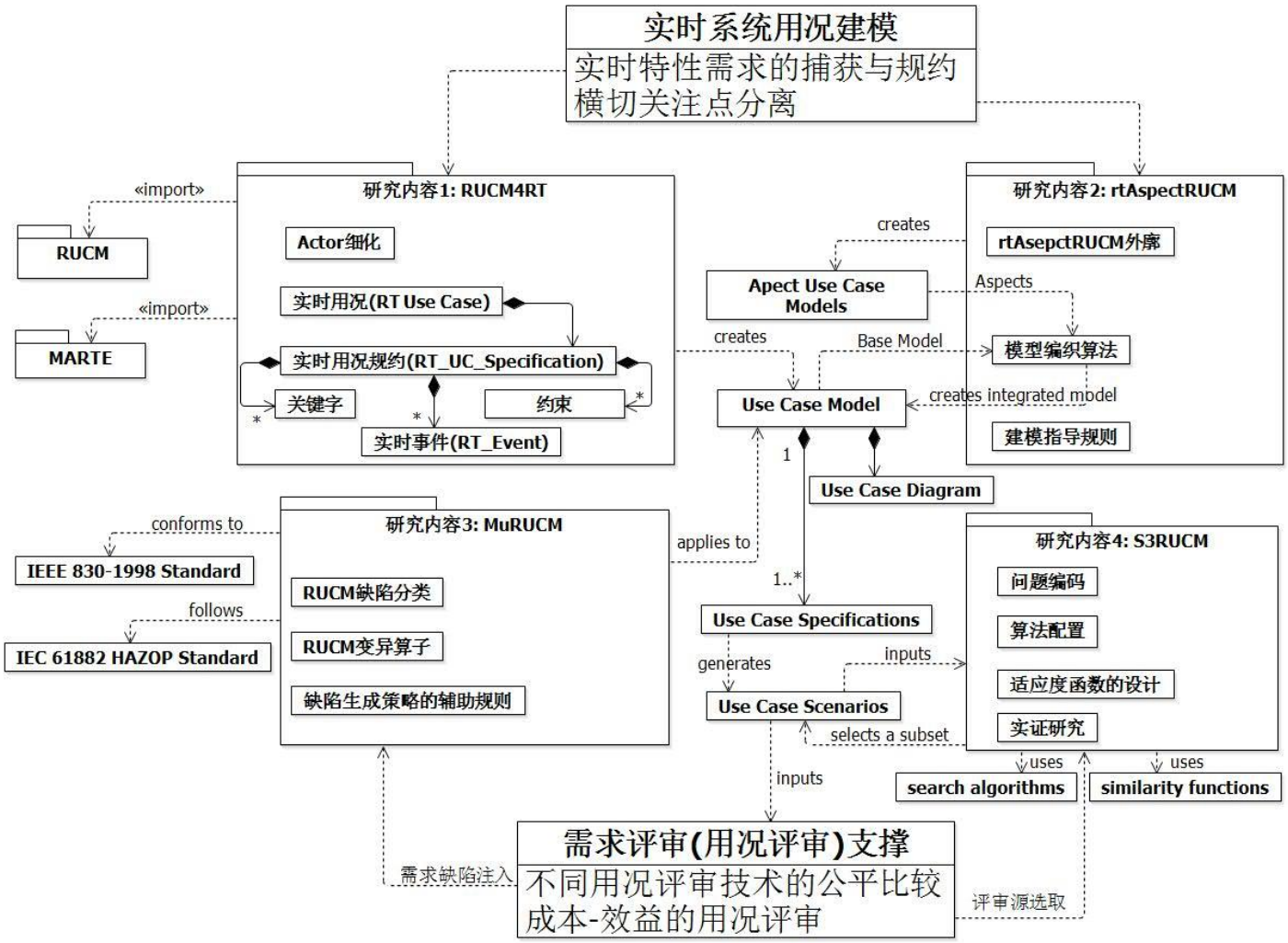

图 3 论文研究框架(使用 UML 类图模型)

1) 基于 RUCM 的实时系统用况建模方法。基于 RUCM 实现对实时系统的用况建模方法(RUCM4RT)。首先对 MARTE[71]外廓(UML profile for Modeling and Analysis of Real-Time and Embedded Systems)进行系统化的研究，MARTE 是目前 UML 对实时系统进行建模的最新指导规范，它提供了丰富的建模概念，特别是那些来自工业领域的反馈和建议。然而 MARTE 并没有针对实时系统的需求建模给出特别的指导，因此首先要研究 MARTE 并识别出那些需要在需求开发阶段就要进行捕获的建模概念，从而成为设计 RUCM4RT 元模型的依据。然后研究 RUCM 的扩展机制，通过 MOF 元模型的建模方式，

分别从参与者 Actor、用况 Use Case、用况规约 Use Case Specificaiton 对新引入的实时性需求相关的概念进行元模型建模。最后研究实时用况场景与 UML 定时图(Timing Diagram)的模型转换规则，实现 UML 定时图的自动生成。

2) 面向实时特性需求的用况建模方法。面对复杂实时系统，为了提高用况模型的可重用、可维护和需求建模的效率，采用面向方面的建模方式实现 RUCM4RT 对横切关注点的分离。首先研究面向方面建模方法的核心概念，如连接点(Jointpoint)，切入点(Cutpoint)，“方面”(Aspect)等。其次，依据面向方面建模的概念模型，在 RUCM4RT 元模型中设计相应的建模元素(如用况切入点、参与者切入点)，从而创建 rtAspectRUCM 外廓，实现“方面”用况模型的创建。然后，设计模型编织算法实现“方面”用况模型与 RUCM4RT 创建的基用况模型的编织，从而创建“集成式”的用况模型。最后，给出相应的建模指导规则，辅助需求人员有效的使用 rtAspectRUCM 和 RUCM4RT。

3) 基于工业标准的用况变异分析。为了系统化的创建用况需求缺陷，首先学习工业标准 IEEE Std. 830-1998[72]并将其应用到 RUCM/RUCM4RT 模型中，从而定义出 RUCM 需求缺陷分类。其次，使用工业标准 IEC 61882 HAZOP[73]对 RUCM 建模元素进行系统分析，从而定义出 RUCM 变异算子，给出需求缺陷的创建操作。最后，给出一系列指导规则用以辅助创建缺陷生成策略。通过这一系列研究实现用况变异分析方法 MuRUCM，为客观公正的评估不同的评审方法创建统一的需求缺陷实例。

4) 基于相似度函数和搜索算法的用况场景选择方法。在评审资源(人力、时间)受限下，面对大量用况场景，采用基于搜索算法的方法择优选择出一部分用况场景作为评审源为需求评审的有效进行提供支撑。首先将用况场景选择的问题转化为一个搜索优化问题并进行问题编码和形式化定义。其次，依据定义好的搜索问题设计相应的适应度函数(fitness function)以指导搜索算法进行有效搜索。然后，选取搜索算法和相似度函数，并进行相应的参数设置。最后，对基于相似度函数和搜索算法的选择策略 S^3RUCM 进行实证研究(Empirical Study)。实证研究的过程严格遵循实证软件工程(Emprical Software Engineering)的实验教程[74]和指导规则[75,76]进行，包括实验设计、实验执行、数据收集、实验分析、风险分析等。针对 S^3RUCM 设计一系列实验研究问题，对收集的实验数据采用相应的统计分析方法(如 Vargha-Delaney[77]统计方法、Kruskal-Wallis[78]秩检验、Wilcoxon[79]秩和检验)进行统计分析，从而给出可信赖的实验结论。

此外，以上四方面的研究内容相互联系，构成有机整体。针对实时系统的用况建模：研究内容(1) RUCM4RT 方法侧重于对实时系统中的实时特性需求、资源约束进行捕获和规约，详细的论述见第 2 章；研究内容(2) rtAspectRUCM 方法着力于采用面向方面的建模方式，实现 RUCM4RT 对横切关注点的分离，从而提高工业实践中用况建模的效率。详细的论述见第 3 章。针对实时系统的用况评审：研究内容(3) MuRUCM 方法研究如何有效的创建 RUCM 用况缺陷，详细的论述见第 4 章；研究内容(4) S^3RUCM 方法研究如何采用成本-效益的方式选择有效的用况场景，从而有效支持用况评审。详细的论述见第 5 章。

本研究中，以上各项研究内容都会采用代表性的工业案例进行验证评估。

1.5 创新点

本文的主要特色和研究创新如下：

1) 提出一种基于 RUCM 的实时系统用况建模方法

RUCM 用况建模方法具有优秀的用况规约能力，通过结构化的用况规约模板和 26 条自然语言限制规则，最大程度上减少需求描述的模糊性和二义性。此外，UCMeta 形式化机制也使得 RUCM 能够更好对用况模型进行分析和转换。MARTE 外廓是最新的实时系统建模指导，包含了丰富的建模概念，通过对 MARTE 的概念提取并引入到 RUCM 中，实现 RUCM4RT 建模方法，辅助需求人员在实时系统的需求开发阶段创建高质量的用况模型和用况规约。

2) 提出一种面向实时特性需求的用况建模方法

面对复杂实时系统，采用面向方面的建模方法，实现对实时系统需求建模中的横切关注点(如资源约束、实时约束)分离。采用 MOF 的元模型建模方式，实现 rtAspectRUCM 外廓，支持对实时性需求的独立建模和维护。rtAspectRUCM 与 RUCM4RT 结合使用，提高了用况模型的可重用、可维护同时也提高了用况建模的效率。

3) 提出一种基于工业标准的用况分析方法

需求评审是需求质量保证的一种有效技术。研究学者提出了不同的评审技术，然而针对它们的有效性评估，研究者却给出了不一致的结论[19]。在这些实证研究中，缺少一种系统化的故障注入方法进行需求缺陷的创建，相应的实验也难以重现。本文从支撑客

观公正的评估不同评审方法的角度出发，依据工业标准 IEEE Std. 830-1998[72]和 IEC 61882 HAZOP[73]定义了全面精确的需求缺陷分类和相应的变异算子，从而可以系统化的创建不同的需求缺陷实例，支持对不同评审方法的客观公正的评估。

3) 提出一种基于相似度函数和搜索算法的用况场景选择方法

在复杂实时系统的工业实践中，一个用况可以诱发大量用况场景的产生，特别是当系统的用况模型中使用了大量的包含«Include»、扩展«ExtendIn» 关系，这种现象在安全关键软件(Safety-Critical Sytem)的开发过程中尤为普遍。在市场驱动开发(Market-driven Development)的背景下，通常无法在有限的成本(时间和人力)内对所有的需求描述完成需求评审，常用的技术手段就是选择部分需求项进行评审[80]，而选择的标准或者是依据领域专家的经验或者是采取随机的策略。本文提出一种自动化的基于相似度函数和搜索算法的方法以选择出一部分用况场景，从而为成本-效益的用况评审的开展提供支撑。

1.6 论文组织结构

第 1 章 绪论。介绍本文的研究背景、相关的研究工作、本文的研究目标和主要研究内容，并说明论文的组织结构。

第 2 章 基于 RUCM4RT 的实时系统用况建模方法。本章提出了一种实时系统的用况建模方法 RUCMRT，详细论述了参与者的细化，实时用况的设计，实时用况规约的设计，并采用工业案例和文献案例对 RUCM4RT 进行验证。

第 3 章 面向实时特性需求的用况建模方法。本章首先根据面向方面建模的概念模型创建了 rtAspectRUCM 外廓，从而支持对“方面”用况模型的创建，然后设计相应的模型编织算法，实现“方面”用况模型与基用况模型的编织融合，最后论述了建模指导规则。本章采用工业案例和文献案例对 rtAspectRUCM 进行验证。

第 4 章 基于工业标准的用况分析方法。本章首先依据工业标准 IEEE Std. 830-1998[72]定义了用况缺陷类型，然后依据工业标准 IEC 61882 HAZOP[73]创建了用况变异算子，最后给出了辅助规则用以创建不同的缺陷生成策略。本章采用工业案例和文献案例对提出的用况变异分析方法进行验证。

第 5 章 基于相似函数和搜索算法的用况场景选择方法。本章首先阐述了 S^3RUCM 用况场景选择方法，然后对用况场景选择问题进行定义，并设计相应的适应度函数，最

后采用实证研究的方法对 S^3RUCM 进行全面综合的验证评估。

最后一章进行全文总结，并指明相应的研究方向和后续研究工作。

第2章 基于 RUCM 的实时系统用况建模方法

2.1 研究问题的提出

在实时系统的设计中，时间相关的属性是一类关键的非功能需求。从实时系统的开发成本、产品质量、生产效率的角度出发，在需求规约和分析阶段，对这些时间相关的属性进行建模和验证，对于实时系统的成功开发是非常重要的。无论是学术研究还是工业实践中，通常采用时序分析技术(例如，最坏执行时间 Worst Case Execution Time)对实时系统的设计进行分析验证，以确保系统设计完全符合相应的时间性能约束。然而这些实时分析技术通常都是在系统设计和开发阶段才可使用，而不是在需求层次进行开展。

用况(Use Case)建模，作为一种对需求进行捕获、描述、分析的建模技术已在工业实践中得到广泛应用[5,6]。用况建模是面向对象(Object-oriented)开发过程，即“以用况为驱动的，以体系架构为中心的，增量、迭代的开发过程”，的核心[4]。在用况建模的工业实践中，Dr. Tao Yue 提出的 RUCM[7](Restricted Use Case Modeling)用况建模方法表现出显著性的优势：消除自然语言的二义性、模糊性[7]，从用况模型到 UML 分析模型(如，类图)的自动转换[8]，测试用例的自动生成[9]。RUCM 用况建模方法是一个通用的用况建模框架，它通过提供良好的扩展机制以支持面向不同应用领域的 RUCM 扩展。因此，可以对 RUCM 进行扩展支持对实时需求的用况建模。

在进行综合模块化航电系统(Integrated Modular Avionics (IMA) systems)这类典型实时系统的开发过程中，工业实践者严格遵循相关的工业标准 DO-178C[2]、DO-332[3]，采用面向对象和模型驱动的技术。这类 IMA 系统通常由一系列计算模块，不同类型的硬件设备，系统与参与者、系统与其他系统或外部系统之间复杂的通讯所构成。在这类 IMA 系统的需求开发过程中，除了复杂的系统功能需求外，还有大量的实时约束、资源限制、通讯协议和异常处理需要被准确捕获并进行精准描述。工业标准 DO-332[3]针对如何使用面向对象技术(Object-Oriented Technology (OOT))及其相关技术进行航空系统的开发给出了详细的指导规则。因此，用况建模技术是我们的工业合作伙伴进行需求开发的工业实践技术。

综上所述，解决工业实践中采用用况建模技术进行 IMA 这类典型实时系统的需求开发，是本章研究问题的现实依据；RUCM 用况建模方法及其良好的扩展机制，为解决该

工业实践提供了技术依据。本章针对实时系统的需求建模，采用限制性自然语言的需求描述方式和元模型(meta-model)的半形式化方式，并基于 UML 关于实时嵌入式系统建模和分析的 MARTE[71]外廓(UML profile for Modeling and Analysis of Real-Time and Embedded Systems)中的相关概念，提出一种实时系统的用况建模方法 RUCM4RT。 此外，通过使用元模型这中形式化的方式，RUCM4RT 支持自动生成各种含有时间约束的用况场景，并针对这些含有实时约束的用况场景自动转换为 UML 定时图(Timing Diagram)。

2.2 RUCM4RT: 实时系统的一种用况建模方法

本节主要对基于 RUCM 用况建模的实时需求建模方法 RUCM4RT 进行详细的阐述。2.2.1 节对 RUCM4RT 的元模型 UCMeta4RT 进行详细论述；2.2.2 节论述了模型转换的工作，即如何从 RUCM4RT 产生的用况场景自动生成 UML 定时图(Timing Diagram)；2.2.3 展示 RUCM4RT 的自动化支持工作。为了对 RUCM4RT 方法进行系统化的论述，图 4 和图 5 展示了一个采用 RUCM4RT 进行用况描述的样例。

RUCM4RT Use Case Specification

Use Case Name	Synchronize with AutopilotSystemB
Brief Description	the system synchronizes with AutopilotSystemB
Precondition	the system starts successfully
Dependency	INCLUDE USE CASE Handle Faults
Generalization	None
Primary Actor	<<Timer>> MainTimer
Secondary Actors	<<ExternalSystem>> AutopilotSystemB
Resources	<<CommunicationMedia>> CCDL
Period	20.0 ms
Time Cost	(120.0, us)---(480.0 us)

Basic Flow	Steps	
(Untitled) ▼	1	MainTimer sends a pulse to activate the system
	2	the system closes external interrupt
	3	the system sends synchronization request data to AutopilotSystemB
	4	the system receives synchronization response data from AutopilotSystemB
	5	the system VALIDATES THAT CONSTRAINT rtDConstraint_1 IS SATISFIED
	6	the system VALIDATES THAT the response data is correct
	7	the system EXECUTES CONCURRENT FLOWS ALT_analog, ALT_discrete, ALT_digital
	8	the system synchronizes with AutopilotSystemB VIA CCDL COMMUNICATION MEDIA
	9	the system opens external interrupt
	Postcondition	the system successfully synchronizes with AutopilotSystemB

图 4 采用 RUCM4RT 的用况规约(基本流)

Specific Alternative Flow "responseError"	RFS 6
	1 the system gets wrong response from AutoPilotSystemB
	2 the system repeates TIMEDPROCESSING BasicFlow steps:3-4 twice
	3 the system VALIDATES THAT rtDConstraint_2 is statisfied
	4 the system VALIDATES THAT the TIMEDPROCESSING BasicFlow steps:3-4 is successful
	5 RESUME STEP 8
	Postcondition the system successfully synchronizes with AutopilotSystemB

Concurrent Alternative Flow "ALT_discrete"	RFS 7
	1 the system sends discrete data to AutopilotSystemB VIA CCDL COMMUNICATION MEDIA
	2 the system receives response data WITHIN 120us
	3 the system VALIDATES THAT response data is valid
	4 RESUME STEP 8
	Postcondition the system finishes synchronization using discrete data

Concurrent Alternative Flow "ALT_analog"	RFS 7
	1 the system sends analog data to AutopilotSystemB VIA CCDL COMMUNICATION MEDIA
	2 the system receives response data WITHIN 120us
	3 the system VALIDATES THAT response data is valid
	4 RESUME STEP 8
	Postcondition the system finishes synchronization using analog data

Concurrent Alternative Flow "ALT_digital"	RFS 7
	1 the system sends digitial data to AutopilotSystemB VIA CCDL COMMUNICATION MEDIA
	2 the system receives response data WITHIN 120us
	3 the system VALIDATES THAT response data is valid
	4 RESUME STEP 8
	Postcondition the system finishes synchronization using digital data

Observation Variables		
@t1	RFS BasicFlow 3	TimeInstant observation of sending synchronization signal
@t2	RFS BasicFlow 4	TimeInstant observation of receiving response
&d1	RFS responseError 2	TimeDuration observation of synchronization error
&d2	RFS ALT_discrete 2	TimeDuration observation of discrete data synchronization
&d3	RFS ALT_analog 2	TimeDuration observation of analog data synchronization
&d4	RFS ALT_digital 2	TimeDuration observation of digital data synchronization

Time Constraints		
rtDConstraint_1	RFS BasicFlow_{3,4}	(0.0, us) < (t2-t1) <= (120.0, us)
rtDConstraint_2	RFS responseError_{2}	(120, us) < d1 <= (240.0, us)
rtDConstraint_3	RFS ALT_discrete_{2}	(0.0, us) < d2 <= (120.0, us)
rtDConstraint_4	RFS ALT_analog_{2}	(0.0, us) <= d3 <= (120.0, us)
rtDConstraint_5	RFS ALT_digital_{2}	(0.0, us) <= d4 <= (120.0, us)

Communication Constraints		
analog data	RFS ALT_analog_{1}	data precision is 12
digital data	RFS ALT_digital_{1}	data transmission rate is (12.5, kb_per_s)

图 5 采用 RUCM4RT 的用况规约(备选流)

2.2.1 UCMeta4RT 元模型

RUCM4RT 的设计目标是在软件需求阶段，针对实时系统的时间特性约束，对其进行捕获并建模为用况模型的一部分。RUCM4RT 继承自 RUCM，采用结构化的需求描述模板和相应的自然语言限制规则，将系统业务中的一系列事件流描述为基本流(basic flow of events)和备选流(alternative flows of events)。针对实时系统的需求特性，RUCM4RT

对原有的 RUCM 需求模板进行扩展，引入新的需求描述项：用况使用资源、用况执行周期、用况执行时间，并提供相应的元模型半形式化机制对时间约束、资源约束进行建模。

本章下面的章节将详细论述 RUCM4RT 的特性和元模型 UCMeta4RT。图 6-图 9 以类图的形式对 UCMeta4RT 进行展示，其中每个类都会通过衍型«RUCM4RT»、«UML»、«RUCM»、 «MARTE»进行标记分类，它们分别表示 RUCM4RT 中新引入的概念、UML 中的元-类型、RUCM 中提出的概念、从 MARTE[71]外廓中借鉴来的概念。

2.2.1.1 Actor 元模型

RUCM4RT 针对实时系统，从用况建模的角度出发，对用况的参与者(Actor)进行细化和分类。如图 6 所示，RUCM4RT 中的 Actor 继承自 UML 的 Actor 和 RUCM 的 Actor(即，PrimaryActor 和 SecondaryActory)。其中，首要参与者(PrimaryActor)代表启动用况的一类角色，而辅助参与者(SecondaryActory)则表示参与用况的交互但不会主动发起交互事件的那一类角色。这种对参与者进行明确划分对实时系统的需求建模是十分必要的，因为在现实中，实时系统处理的业务有些是参与者主动发起的，同时实时系统也会主动向参与者发起交互，例如，系统进行异常事件的处理。更进一步，RUCM4RT 中的参与者又细化为：人参与者(HumanActor)、外系统参与者(ExternalSystem)、资源参与者(ResourceActor)。实时系统的业务流程中不仅会有人的参与，同时也会使用各种不同的资源与不同的系统设备进行交互。因此，RUCM4RT 希望通过对参与者进行资源、外部系统、参与人的明确细化，以方便需求人员进行用况建模时，可以清楚的界定系统边界，明确系统的业务。其中资源参与者(ResourceActor)又细化为：存储资源参与者(StorageResourceActor)、通讯资源参与者(CommunicationResourceActor)、设备资源参与者(DeviceResourceActor)、计算资源参与者(ComputingResourceActor)和定时参与者(Timer)。相比较一般系统的设计，资源的使用对于实时系统的设计尤为重要。实时系统通常会进行嵌入式设计，在这种架构下系统可以使用的资源受外部环境和设备物理特性的限制，往往要求系统必须要高效率的使用各种资源。因此，对实时系统中的资源参与者在需求开发阶段进行详细建模可以为后续进行的系统设计提供直接依据和支持。如图 4 所示，用况‘Synchronize with AutopilotSystemB’ 的首要参与者 MainTimer 被建模为一种定时参与者(Timer)；此外，该

用况是从 *AutopilotSystemA* 系统的角度进行的建模，因此将对等系统 AutopilotSystemB 建模为一种外系统参与者(`ExternalSystem`)。

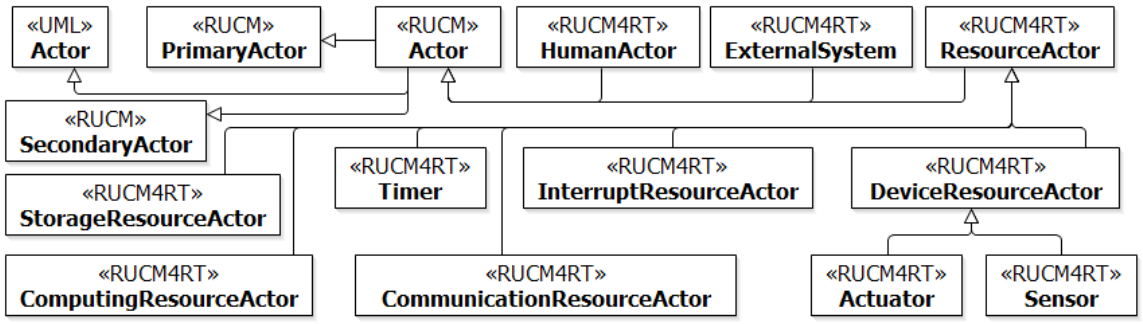

图 6 参与者 Actor 的元模型

2.2.1.2 用况(Use Case)和用况规约(Use Case Specification)的元模型

MARTE 外廓通过引入相关的概念(即，周期模式 PeriodicPattern、迸发 BurstPattern、不规则模式 IrregularPattern、零星模式 SporadicPattern)对实时系统中的事件的不同到达方式进行了建模。如同实时系统设计著作[40]中论述的，系统的动作行为最终都是由事件激励触发的，因此，RUCM4RT 将 MARTE 中提出的这些概念引入到用况建模中用以创建不同类型的用况。如图 7 的用况元模型所示，RUCM4RT 中将实时用况分为两种类型：非周期性实时用况(AperiodicRTUC)和周期性实时用况(PeriodicRTUC)。一个周期性实时用况(PeriodicRTUC)是被周期性触发启动的，它用来捕获那些周期性的任务并对其进行建模描述。例如，图 4 所示，用况‘Synchronize with AutopilotSystemB’就是一个周期性的用况。非周期性实时用况(AperiodicRTUC)用来对那些非周期性的实时行为进行建模。非周期性实时用况(AperiodicRTUC)又进一步细化为：不规则实时用况(IrregularRTUC)、零星实时用况(SporadicRTUC)、迸发实时用况(BurstRTUC)，它们分别对应不同的事件到达模式。不规则实时用况(IrregularRTUC)，其系统动作行为可以通过为一系列连续的时间间隔进行抽象描述[71]。零星实时用况(SporadicRTUC)，系统的动作行为对应着零星模式的时间处理，即这些事件的发生比较琐碎没有固定的时间规律，只能通过一个最小时间间隔和最大时间间隔进行抽象描述[71]。迸发实时用况(BurstRTUC)，系统的动作行为可以描述为：在一个给定的迸发时间间隔内，完成最大数量的迸发事件的处理[71]。 例

如，“一个按钮操作的需求可能被描述为：在 20ms 内完成 10 次的单击操作[40]”。

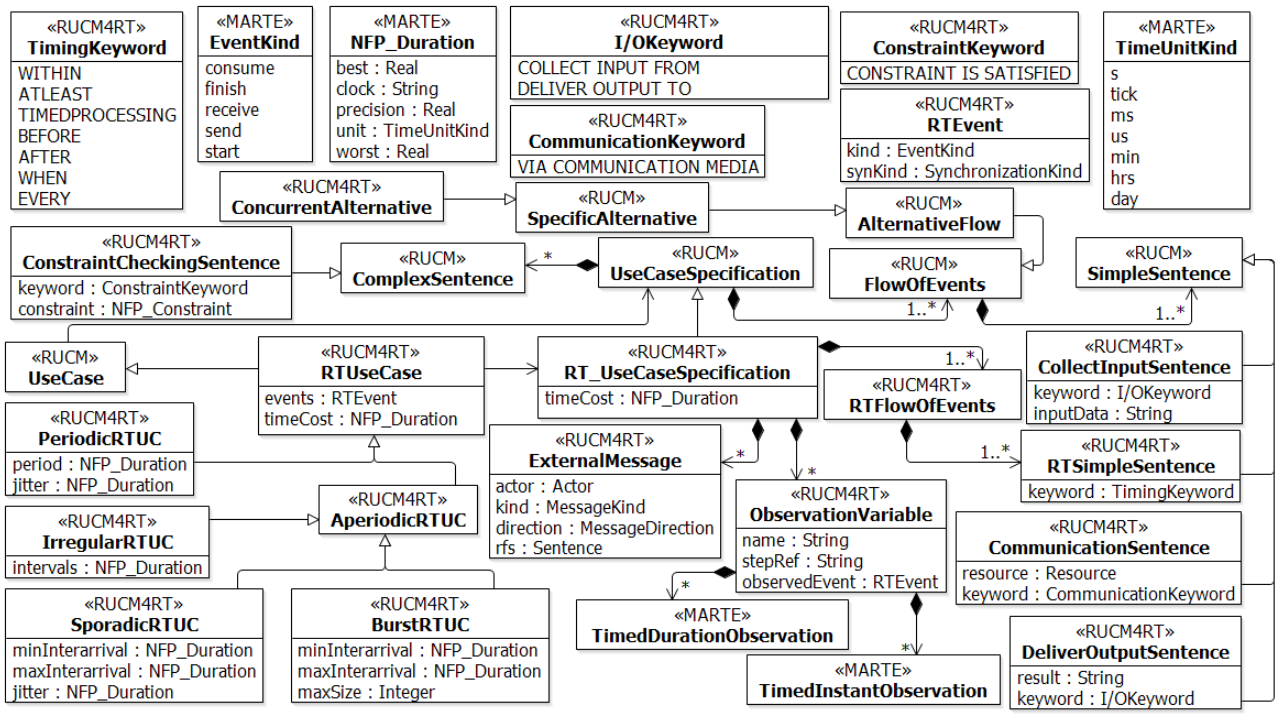

图 7 用况和用况规约的元模型

实时用况规约模板(RT_UseCaseSpecification)继承自 RUCM 用况描述模板，除了基本流(basic flow)用于描述系统的主要事件流，还有三种备选流(alternative flow)：特定流(SpecificAlternativeFlow)、组界流(BoundedAlternativeFlow)、全局流(GlobalAlternativeFlow)。此外，在 RUCM4RT 中，还引入了另外一个备选流：并发流(ConcurrentAlternative) 用于描述并发事件。每一个并发流都指向某个事件流中的某个具体的需求描述语句，该需求描述语句是系统产生并发行为的依据，也是各个并发流产生分支的点。在 RUCM4RT 中，设计关键字‘EXECUTES CONCURRENT FLOWS’来明确指明并发流的产生。例如，图 4 所示，基本流中的第 7 个需求描述语句说明系统要执行并发流：‘ALT_analog’、‘ALT_discrete’、‘ALT_digital’。

RUCM4RT 同时还对 RUCM 中的需求描述的简单语句(SimpleSentence[2]) 进行了四种具体的扩展：数据收集语句(CollectInputSentence)、实时简单描述语句(RTSimpleSentence)、通讯 m 描述语句(CommunicationSentence)、结果输出语句(DeliverOutputSentence)。针对以上每一种需求描述语句，都设计了相应的关键字支持。例如，当使用 RUCM4RT 进行

[2]In linguistics, a simple sentence has one independent clause and no dependent clauses: one subject and one predicate.

需求描述时，如果描述语句中使用了预先定义的实时性关键字(如，‘WITHIN’)，这些实时建模描述语句(RTSimpleSentence)就能够被自动化的识别出来。如图 5 所示，在并发流‘ALT_digital’的第 2 个需求描述语句中，使用了‘WITHIN’这个实时需求描述的关键字，即‘the system should receive the response data WITHIN 120μs’。RUCM4RT 中同时还设计了其他的实时关键字：‘BEFORE’、‘WHEN’、‘ATLEAST’，用以在进行自然语言的需求描述时对实时信息进行捕获和描述。相似的，如图 4 所示，通讯需求描述语句(CommunicationSentence)在基本流的第 8 个需求描述语句中进行了使用，其中关键字‘VIA COMMUNICATION MEDIA’用以明确指明系统需要使用交叉通道数据链路(Cross Channel Data Link (CCDL))直线系统之间的同步。通过这样明确的描述，后续在进行系统分析和设计是，可以针对 CCDL 的特性(如，数据传输速度、物理限制)进行科学有效的设计从而为产品的最终质量和开发效率提供支持。RUCM4RT 中同时还针对 CollectInputSentences 和 DeliverOutputSentences 设计了相应的数据输入和输出操作的关键字用以明确的指明系统输入数据的来源和系统输入数据的目的地，从而便于后续的分析和设计。

如图 7 的用况元模型所示，元模型观察变量(ObservationVariable)用于对应着事件的发生。每一个观察变量(ObservationVariable)都有一个类型是 `RTEvent` 的属性‘`observedEvent`’用以指明当前事件的具体类型和实例。RUCM4RT 中的事件类型的元模型 `RTEvent` 又细化为：RTTimedEvent、RTSignalEvent、RTChangeEvent，其中这些事件的分类方式是从 MARTE[71] 中借鉴来的。RTTimedEvent、RTSignalEvent、RTChangeEvent 分别用于对时间事件、信号事件、变化事件进行建模。同时采用 MARTE 中的 TimedInstantObservation 和 TimedDurationObservation 的建模概念将其与观察变量(ObservationVariable)进行关联。TimedInstantObservation 用于对一个时间点的事件进行描述，而 TimedDurationObservation 则用于对时间间隔的事件进行描述。TimedInstantObservation 和 TimedDurationObservation 都有一个属性‘`interval`’，其类型是 NFP_Duration[71]。如图 5 所示，变量@t1 是一个 TimedInstantObservation 实例，它指向基本流中的第 3 个需求描述语句，并暗示：系统在进行基本流中第 3 个需求描述语句时会发生一个 RTSignalEvent 事件发送一个系统同步信号。&d4 是 TimedDurationObservation 的一个实例，它指向并发流‘ALT_digital’(图 5)中的第 2 个需

求描述语句。RUCM4RT 采用 RFS 关键字这种机制可以非常方便的讲一个观察变量与对应的需求描述语句进行关联。

2.2.1.3 约束的元模型

对实时嵌入式系统而言，对资源约束进行需求建模和精确描述是非常重要的。在 RUCM4RT 中，如图 8 所示，针对每个资源约束，都为其定义相应的作用范围(ConstraintScope)、对应的需求描述语句(stepRef:String)、约束表达式(expression)。RUCM4RT 将资源约束(ResourceConstraint)细化为：存贮资源约束(StorageConstraint)、通讯约束(CommunicationConstraint)、设备资源约束(DeviceConstraint)、计算资源约束(ComputingConstraint)、时间约束(TimingConstraint)。这种划分对应于 MARTE 中资源概念的 划分：存贮资源(StorageResource)、通讯资源(CommunicationResource)、设备资源(DeviceResource)、计算资源(ComputingResource)、时间资源(TimeResource)。例如，图 5 所示，并发流‘ALT_digital’中第 1 个需求描述语句，存在一个通讯约束(CommunicationConstraint)：当使用 CCDL 进行数字数据传输时，其数据传输的速率必须是 12.6kb/s。

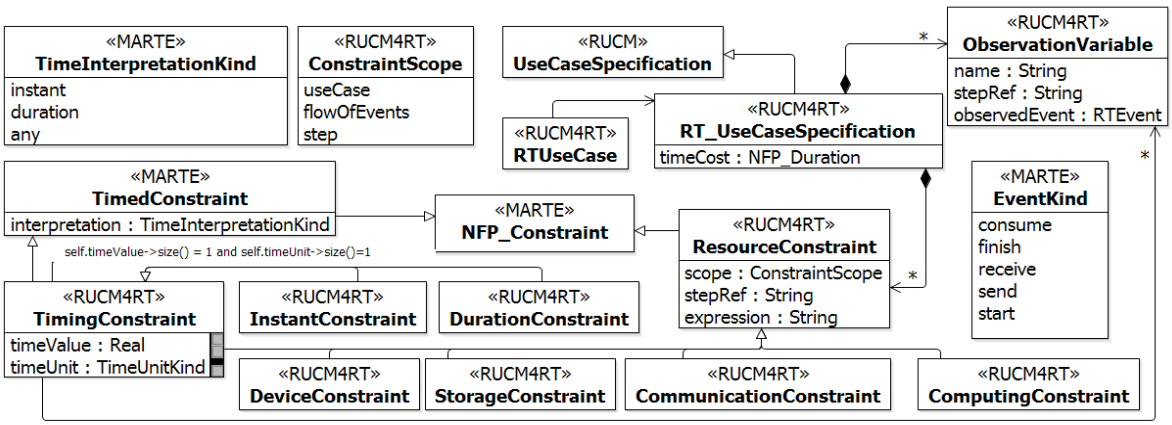

图 8 RUCM4RT 中约束的元模型

RUCM4RT 中的时间约束(TimingConstraint)继承自 MARTE 中的时间约束 TimedConstraint 概念。RUCM4RT 提供两种时间约束的实例：时间点约束(InstantConstraint)和时间段约束(DurationConstraint)。时间点约束(InstantConstraint)，用于对某个具体的时间点进行建模描述，而时间段约束(DurationConstraint)，借助时间单位和时间值对时间间隔进行建模描述。当使用时间约束(TimingConstraint)进行需求建模时，

它可以使用观察变量(ObservationVariable)。例如，图 4 所示，在基本流的第 3 步中 *AutopilotSystemA* 向 *AutopilotSystemB* 发送了一个系统同步的信号，而在基本流的第 4 步中 *AutopilotSystemA* 接收 *AutopilotSystemB* 返回的应答响应；因此，两个观察变量@t1、@t2 被创建来对两个时间的发生进行建模，分别是@t1：发送事件和@t2：接收事件。同时第 3 步和第 4 步的系统动作必须要在 120μs内完成，而这一时间约束就被建模为一个时间段约束(DurationConstraint)，如图 5 所示，(0.0, μs) < t2-t1 <= (120.0, μs)。当进行时间约束的建模时，如图 8 中的 OCL 约束所示，其相应的时间数值(timeValue)和时间单位(timeUnit)必须要进行精确的描述。

2.2.1.4 事件流的元模型

在 RUCM4RT 中，如图 9 所示，原来 RUCM 中的事件流建模元素被 RUCM4RT 进行了扩展，包含了实时事件(RTEvent)的信息。实时事件(RTEvent)继承自 UML 中的元-类型：事件(Event)，它同时还有一个属性 `kind`，其数据类型是 MARTE 中的 `EventKind`。实时事件(RTEvent)又细化为：实时变化事件(RTChangeEvent)、实时信号事件(RTSignalEvent)和时间事件(RTTimedEvent)，分别对应于 UML 中的概念：变化事件(ChangeEvent)、信号事件(SignalEvent)、时间时间(TimeEvent)。这些实时事件都会被 RUCM4RT 中的事件流(RTFlowOfEvents)和观察值(ObservationVariable)使用。例如，图 5 所示，*AutopilotSystemA* 系统在与 *AutopilotSystemB* 进行系统同步的过程中可能需要进行错误处理，一旦进行错误处理，那么必须要 240 μs内完成相应的处理操作。而这个错误处理的时间事件将会被时间段观察变量&d1 使用，最终，时间段观察变量&d1 将用于时

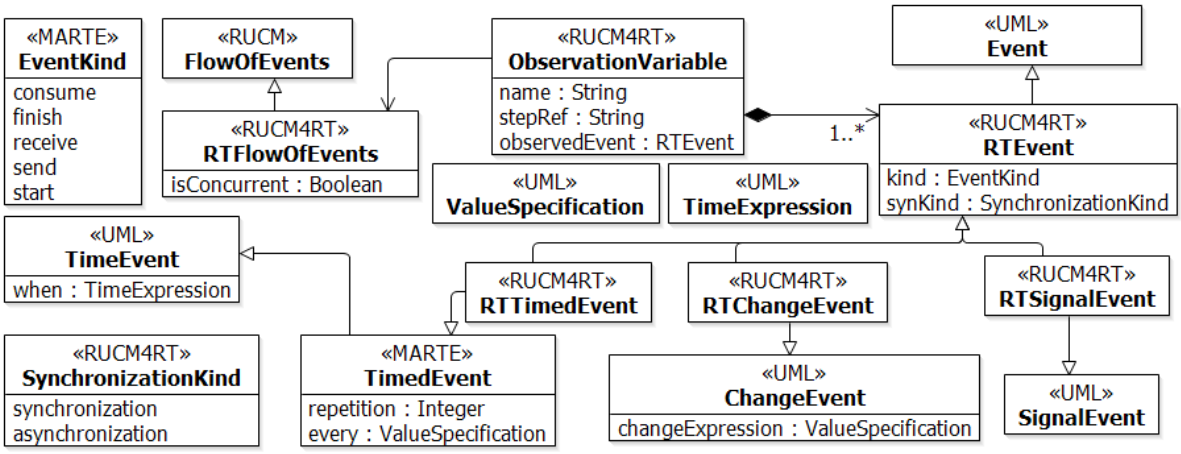

图 9 RUCM4RT 中事件流的元模型

间段约束的建模：‘(120, μs) < d1 <= (240, μs)’。

2.2.2 用况场景到 UML 定时图(Timing Diagram)的模型转换

本节针对 RUCM4RT 用况场景的自动生成和相应的模型转换进行论述。术语“Timing Diagram”在国内有不同的中文翻译，本文撰写中采用北京大学邵维忠老师和杨芙清老师提倡的翻译：“定时图”。在著作《面向对象的系统分析》[4](第 2 版)中 212 页，两位老师给出了详细的论述，感兴趣的读者请进行参阅。

2.2.2.1 RUCM4RT 用况场景的自动生成

定义 2.1：RUCM 用况场景。一个 RUCM 用况场景就是一个封装了一系列顺序执行的 RUCM 用况需求描述语句(RUCM Steps)的序列；它总是从 RUCM 基本流(basic flow)开始，可能会横贯一个或多个 RUCM 备选流(alternative flow)，最后会借助 RFS 关键字返回基本流中结束或者使用 ABORT 关键字在备选流中结束。一个 RUCM 用况通常会产生一组 RUCM 用况场景。

其形式化定义是一个四元组S_{cn}：$\langle BF, AF, RESUME, ABORT\rangle$，其中$BF$是唯一的基本流，$AF$可以是一个或多个备选流，$RESUME$和$ABORT$分别代表 RUCM 中关键字 RESUME STEP 和 ABORT，并且AF的个数是$RESUME$个数和$ABORT$个数的总和。

例如，图 5 所示，并发流‘ALT_digital’是从基本流的第 7 个需求描述语句分支出来的，它对应的用况场景$S_{cn_{ALT_digital}}$就是：Basic flow step 1→ Basic flow step 2→ Basic flow step 3→ Basic flow step 4→ Basic flow step 5→ Basic flow step 6→ ALT_digital step 1→ ALT_digital step 2→ ALT_digital step 3→ Basic flow step 8→ Basic flow step 9。

定义 2.2：RUCM4RT 用况场景。相比较普通的 RUCM 用况场景， 如果一个 RUCM4RT 用况场景是从非实时用况中产生而来的，那么这个 RUCM4RT 用况场景与普通的 RUCM 用况场景一样，如果它是从一个实时用况产生而来的，那么除了具备 RUCM 需求描述语句序列同时还有一组相应的时间相关信息。

其形式化定义是一个五元组RTS_{cn}：$\langle BF, AF, RESUME, ABORT, RTI\rangle$，其中$BF$是唯一的基本流，$AF$可以是一个或多个备选流，$RESUME$和$ABORT$分别代表 RUCM 中关键字 RESUME STEP 和 ABORT，并且AF的个数是$RESUME$个数和$ABORT$个数的总和，

RTI是该实时用况场景中包含的实时信息。

RTI又是一个三元组RTI：$\langle RTEvent, Observationvariable, ResourceConstraint\rangle$

其中，$RTEvent$是该实时用况场景中包含的 RUCM4RT 实时事件，$Observationvariable$是 RUCM4RT 观察变量，$ResourceConstraint$是 RUCM4RT 约束，并且$(\#(RTEvent) + \#(Observationvariable) + \#(ResourceConstraint)) \geq 1$。

每个 RUCM4RT 用况都会被形式化为一个 UCMeta4RT 实例，RUCM4RT 借助自定义的事件流、关键字(如，DO-UNTIL, INCLUDE USE CASE)并采用一种结构化的方法将需求描述中的控制流信息进行捕获建模。本文采用三种不同的结构化覆盖标准：全条件覆盖(*All Condition Coverage*)、全事件流覆盖(*All FlowOfEvents Coverage*)、全语句覆盖(*All Sentence Coverage*)对给定的RUCM4RT用况规约进行RUCM4RT用况场景的自动生成。上述三种覆盖策略已在 RUCM 方法中实现，本文只是采用这些覆盖标准。全条件覆盖(*All Condition Coverage*)确保所有引起分支的 RUCM 条件描述语句都被覆盖掉且被执行至少一次。对于用况描述中的迭代结构体(如，DO-UNTIL)，RUCM 在生成用况场景时都会保证迭代结构体可以执行一次或 x 次，其中 x 是一个可配置的正整数。在本文的实验中，x 被设定为 1，即执行一次。全事件流覆盖(*All FlowOfEvents Coverage*)确保 RUCM 基本流和所有的备选流至少被覆盖掉一次。对于一个特定流(specific flow)或者组界流(bounded flow)，借助于分支指示语句(RFS flow step)，往往会产生一个或多个 RUCM 分支。对于全局流(global flow)， RUCM 会针对每个 RUCM 事件流选择一个描述语句创建相应的分支。全语句覆盖(*All Sentence Coverage*)确保所有的 RUCM 描述语句都被覆盖掉至少一次。 以上覆盖策略已被用于生成测试用例[81,23]。

2.2.2.2 UML 定时图(TimingDiagram)的自动生成

针对每一个包含实时信息的 RUCM4RT 用况场景(*rts*)，RUCM4RT 可以自动化的为 *rts* 创建相应的 UML 定时图(Timing Diagram), 表 5-表 6 展示了相应的转换规则。为了创建 UML 定时图(Timing Diagram)，R0 负责从相应的 RUCM4RT 用况规约中选择出 *rts* 相应的必要模型元素：`ExternalMessages`、`ObsevationVariables`、

表 5 从 RUCM4RT 实时用况场景(*rts*)到 UML 定时图的转换规则(表一)

ID	转换规则描述
R0	针对 *rts* 中包含的每个 RUCM 需求描述语句(step)，从所在的 RUCM4RT 用况规约中选择当前 step 对应的 ExternalMessages 、 ObsevationVariables 、 TimingConstraints 元 素 ， 例 如 ， ExternalMessage.rfs 便指明了相应的 step。
R1	针对每一个 R0 中选择出的 ExternalMessage，为其相应的建模元素 primary actor 和 secondary actor 创建相应的 UML 建模元素:生命线 *Lifeline*。其中 ExternalMessage.actor 指明了相应的参与者 actor。为当前系统创建一个 UML 生命线元素：*TheSystemLifeline*。
R1.1	If (ExternalMessage.actor.typeof(Timer actor), Then 创建一个缺省的 UML 元素：State Invariant 'ticktock'，并将其与 R1 中创建的相应的 Timer 的生命线 lifeline 进行关联。
R1.2	Else 创建一个缺省的 UML 元素：State Invariant 'waiting'，并将其与 R1 中创建的相应的 Timer 的生命线 lifeline 进行关联。
R2	针对 *rts* 中的每一个 step，为其场景一个相应的 UML 元素：State Invariant，并将其与系统生命线 *TheSystemLifeline* 进行关联。
R3	针对 R0 中识别出的每一个 ExternalMessage，将其创建为一个 UML 元素：Message。 ExternalMessage.actor ：相应的 actor 生命线； ExternalMessage.rfs：系统生命线 *TheSystemLifeline* 中对应的 State Invariant。 ExternalMessage.kind：消息类型，如，同步、异步。
R3.1	If (ExternalMessage.actor.typeof(Timer actor)，Then 创建一个 UML 元素:State Invariant 'sending message'，并将其与相应的 Timer actor 生命线进行关联。

R3.2	Else
	If (ExternalMessage.direction is in)，创建一个 UML 元素：State Invariant 'sending message'，并将其与相应的 Timer actor 生命线进行关联。
	If (ExternalMessage.direction is out)，创建一个 UML 元素：State Invariant 'receiving message' ，并将其与相应的 Timer actor 生命线进行关联。

R4	针对每一个 *rts* 中的 TimedInstantObservation，创建一个 UML 元素：Time Observation TimedInstandObservation.ref：系统生命线 *TheSystemLifeline.*中相应的 State Invariant； 设置 TimeObservation.setEvent(State Invariant).

表 6 从 RUCM4RT 实时用况场景(*rts*)到 UML 定时图的转换规则(表二)

ID	转换规则描述
R5	将 *rts* 中的每一个 TimedDurationObservation 转换成 UML 定时图中的 Duration Observation： 1) TimedDurationObservation.ref：系统生命线 *TheSystemLifeline* 中的所有 State Invariants； 2) 针对 TimedDurationObservation.ref 中的每一个 State Invariant 创建 UML 元素：Duration Observation，并进行设置：DurationObservation.getEvents().add(State Invariant).
R6	将 *rts* 中的每一个 InstantConstraint 转换成 UML 定时图中的 Time Constraint： 1) InstantConstraint.rfs：系统生命线 *TheSystemLifeline* 中对应的 State Invariant； 2) 创建一个 UML 元素：Time Constraint 和一个 UML 元素：Time Interval； 3)进行设置： TimeInterval.setMax(InstantConstraint.interval.max)， TimeInterval.setMin(InstantConstraint.interval.min)， TimeConstraint.setSpecification(TimeInterval)。
R7	将 *rts* 中的每一个 DurationConstraint 转换成 UML 定时图中的 Duration Constraint： 1) 创建一个 UML 元素：Duration Constraint 和一个 UML 元素：Duration Interval； 2) 进行设置： DurationInterval.setMax(DurationConstraint.max)， DurationInterval.setMin(DurationConstraint.min)， DurationConstraint.setSpecification(DurationInterval)。
R8	针对 Timer actor 生命线，为其创建一个缺省的 State Invariant ‘ticktock’； 针对其他的 actor 生命线，为其创建缺省的 State Invariant ‘waiting’。

SecondaryActor 会被自动化的转换成 UML 中的建模元素，即元-类型：Lifeline 同时也会为当前系统创建相应的生命线‘*TheSystemLifeline*’。 在转换规则 R2 中，UML 建模元素：State Invariants 会从 *rts* 中进行创建，并将它们与系统生命线‘*TheSystemLifeline*’进行关联。针对信息 Message 的转换，在当前的实现中，RUCM4RT 主要考虑那些系统与参与者之间的交互信息，并提供元模型 ExternalMessage (图 7)对其进行建模。

R3 描述了 UCMeta4RT 中的元模型 ExternalMessage 与 UML 中的元模型 Message 直接的转换映射关系。R3 又具体细分为两个子转换规则,其中 R3.1 主要负责识别消息关联的两端对象，而 R3.1 用于识别消息的类型(如，同步、异步)和消息的方向。UML 定时图(Timing Diagram)中的其他建模元素由转换规则 R4-R8 负责转换. 例如，R7 负责创建 UML 中的建模元素：DurationConstraint。如图 8 所示，UCMeta4RT 的元模型 TimeDurationConstraint 的属性‘*ref*’ 便指向，针对当前用况所对应的当前系统，所创建的生命线 Lifeline 中的 State Invariant ，而 DurationConstraint 的属性‘min’和‘max’ 将会分别被转换成 UML 定时图(Timing Diagram)中的 Duration 的最小时间值(minimum)和最大时间值(maximum)。类似的，分别依据转换规则 R4、R5、R6，UML 定时图(Timing Diagram)中的建模元素：TimeObservation、DurationObservation、TimeConstraint 也会自动从 UCMeta4RT 中对应的元-类型：TimeInstantObservation 、 TimeDurationObservation(图 7) 和 InstantConstraint(图 8)。最后，R8 负责为 UML 定时图(Timing Diagram)中的每个参与者(Actor)生命线创建其缺省的 State Invariant。在本文中，并不关心每个参与者(Actor)的具体业务状态，因此 RUCM4RT 中采用‘ticktock’ 作为时间参与者 Timer 的缺省状态，并使用‘waiting’作为其他非时间参与者的缺省状态。

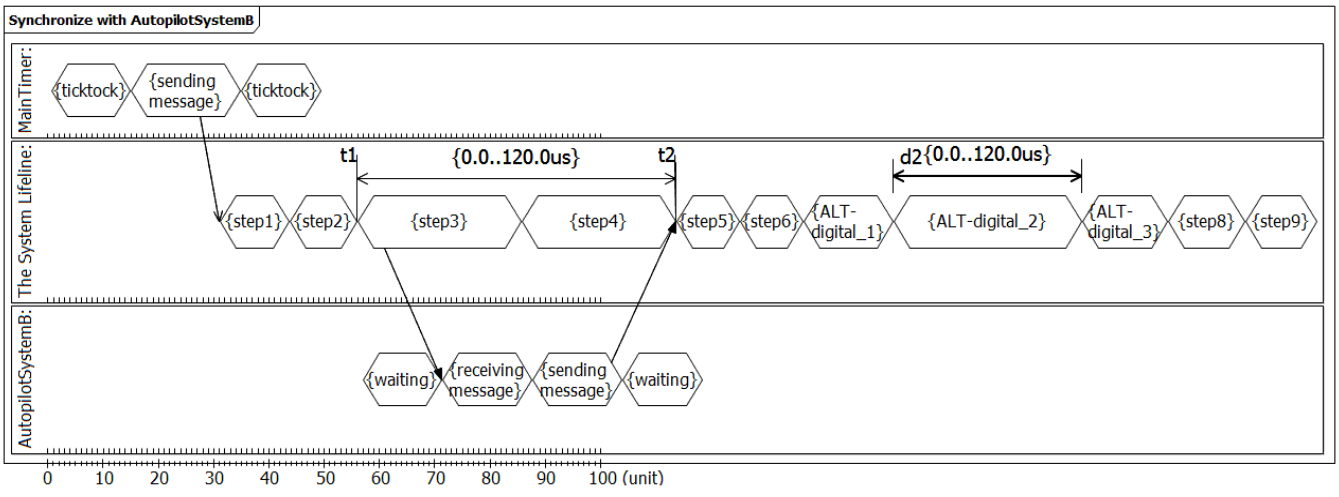

图 10 一个自动生成的 UML 定时图(Timing Diagram)

图 10 展示了由‘ALT_digital’(图 5)对应的实时用况场景$RTS_{cn_{ALT_digital}}$(2.2.2.1 节)自动生成的 UML 定时图(Timing Diagram)。如图 10 所示，针对实时用况场景$RTS_{cn_{ALT_digital}}$

其所有相关的实时信息，如 Duration Constraint、Time Observation，都已被图形化显示。例如，时间约束‘(0.0, μs) < d4 <= (120.0, μs)’(图 5)会被形式化为一个 DurationConstraint 实例，其最小值是(0.0, μs) 最大值是(120.0, μs)。通过这样做，需求人员可以针对某个具体的 RUCM4RT 场景，借助这些图形化显示的标记对其进行相应的分析。目前的 RUCM4RT 自动化工具(2.2.3 节)还不支持 UML 定时图(Timing Diagram)的自动显示化，它必须要借助外部工具 IBM RSA [82])的支持来实现。

2.2.3 自动化支持

Zen-RUCM[83]提供相应的扩展机制，以方便针对具体的应用领域进行基于 RUCM 的建模工具的实现。本章中 RUCM4RT 的自动化工具就是基于 Zen-RUCM[83]实现的。图 11 和图 12 展示了 RUCM4RT 的自动化工具，从中可以观察到，RUCM4RT 自动化工具主要由 5 个编辑器构成：模型树编辑器(Model-Tree editor)、用况规约编辑器(Use Case Specification editor)、建模元素的属性编辑器(Property editor)、用况图编辑器(Use Case Diagram editor)、UML 定时图生成器(timing diagram generator)。这些编辑器都已在图 11 和图 12 进行了相应的标记。

模型树编辑器(Model-Tree editor)主要被设计来方便需求人员创建用况建模元素：用况(Use Case)、参与者(Actor)、时间约束等。即，2.2.1 节中论述的 UCMeta4RT 的各个元模型都会在模型树编辑器(Model-Tree editor)中被实现。该编辑器也是最主要的一个用户接口。模型树编辑器(Model-Tree editor)通常会搭配建模元素的属性编辑器(Property editor)一起使用，用户使用建模元素的属性编辑器(Property editor)针对模型树编辑器(Model-Tree editor)创建的不同的元素为其指定相应的属性值。例如，图 5 所示的时间约束‘(120, μs) < d1 <=(120, μs)’，首先会在模型树编辑器(Model-Tree editor)中创建时间段约束 DurationConstraint 的一个实例 d1，然后借助属性编辑器(Property editor)，设置 d1 的最小值(120, μs)和最大值(120, μs)。

用况规约编辑器(Use Case Specification editor)实现了 RUCM4RT 的用况模板，即 2.2.1 节中论述的 Uase Case Specification 的元模型。用况规约编辑器(Use Case Specification editor)方便用户针对某个具体的 RUCM4RT 用况进行需求描述，如图 4-图 5 所示，用况的所有信息，如前置条件、首要参与者、辅助参与者、依赖关系(«Include»、«Extend»、基本流(basic flow)、备选流(alternative flow)，都需要需求人员在用况规约编辑器(Use Case

Specification editor)中进行详细准确的描述。所有这些文本描述的需求内容(RUCM step)和各个用况描述信息(如前置条件、首要参与者、辅助参与者)都会被实例为相应的 UCMeta4RT 元模型。RUCM4RT 采用 UCMeta4RT 元模型的方式将用况描述信息进行形式化，这样有利于后续的模型分析和转换。如图 4-图 5 所示，预定义的各种 RUCM4RT

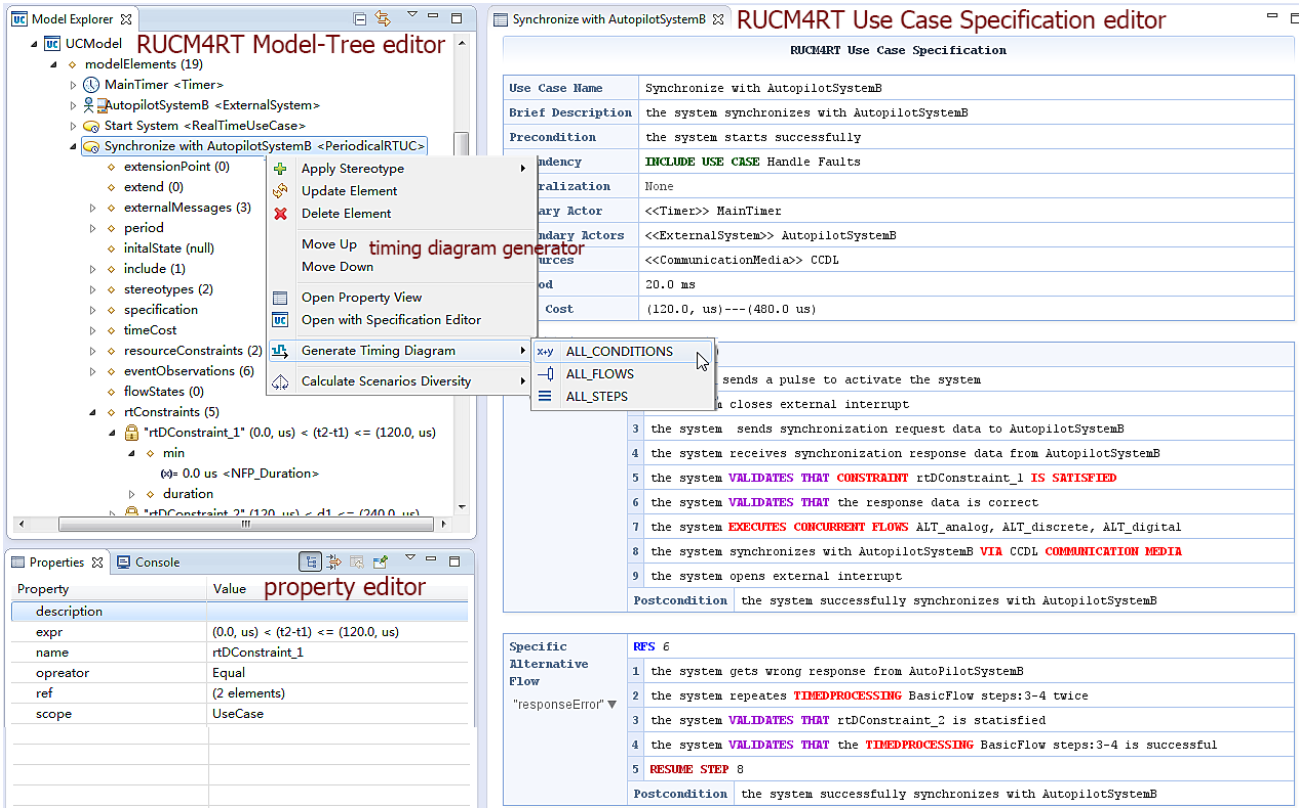

图 11 RUCM4RT 工具(模型树编辑器、用况规约编辑器、Timing Diagram 生成器)

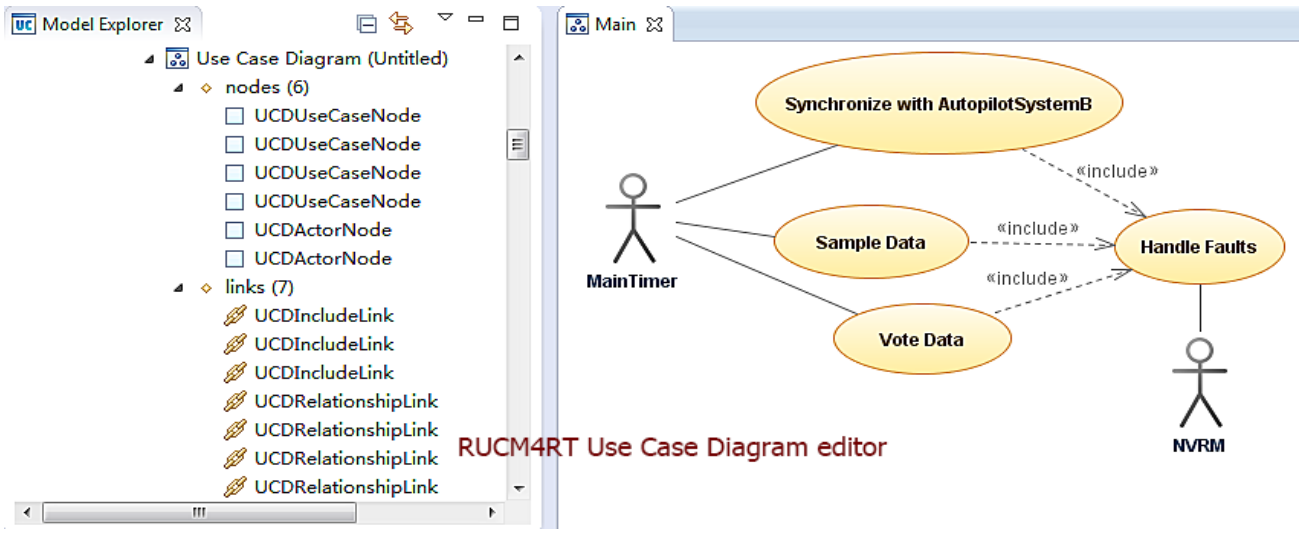

图 12 RUCM4RT 工具(用况图编辑器)

关键字(如‘BEFORE’、‘WHEN’、‘ATLEAST’、 ‘VIA COMMUNICATION MEDIA’)都会在用况规约编辑器(Use Case Specification editor)中被自动识别。

给定一个 RUCM4RT 用况规约，UML 定时图(Timing Diagram)生成器，首先会依据具体的 RUCM 覆盖标准(2.2.2.1 节：全条件覆盖(*All Condition Coverage*)、全事件流覆盖(*All FlowOfEvents Coverage*)、全语句覆盖(*All Sentence Coverage*))自动生成相应的 RUCM4RT 用况场景组，然后针对每一个 RUCM4RT 用况场景都会为其创建相应的 UML 定时图(Timing Diagram)。最后，如图 12 所示，用况图编辑器(Use Case Diagram editor)为需求人员提供一个图形标识的接口，方便需要人员创建相应的用况图模型。

2.3 案例研究

为了对 RUCM4RT 的有效性进行评价，本节分别从航空领域和航海领域选用两个工业案例对 RUCM4RT 的建模能力进行分析。

2.3.1 研究问题

为有效评估 RUCM4RT 对实时系统的需求建模能力，本章设计如下的研究问题：

RQ1：RUCM4RT 是否有效的帮助需求人员针对实时性需求进行用况建模?

RQ2：RUCM4RT 是否有效的生成 UML 定时图(Timing Diagram)?

2.3.2 案例描述

该实验选用了两个工业案例。第一个工业案例是一个飞行控制系统(NAS[102])，它对数据采集器返回的各种飞行参数按照控制律计算公式进行数据计算，并产生相应的飞行控制指令，将飞行控制指令发送给飞行舵机从而控制飞行器的安全飞行。该飞行控制系统(NAS)具有两种飞行操作模式：自动驾驶飞行模式，在该模式下不需要飞行员的飞行指令；人工驾驶飞行模式，该模式需要飞行员的飞行指令。一个飞行员可以切换这两种飞行模式。在进行这种综合模块化航电系统(Integrated Modular Avionics (IMA) systems)的开发过程中，我们的工业合作伙伴严格遵循相关的工业标准如 DO-178C[2]，采用面向对象和模型驱动的技术。由于 NAS 是一个安全关键的系统，所以其设计采用双系统冗余的设计方案。

该实验中从 NAS 系统选取了 13 个用况对其进行建模描述。下面对其中 11 个典型

的实时用况进行说明。启动系统 Start System (UC1)，用于完成系统启动时的初始化相关操作；上电自检 Power-up Built-in Test (UC2)，主要负责系统上电后的一系列设备、部件检查工作；故障处理 Handle Faults (UC3) 主要被设计来解决系统运行中出现的各种错误和异常；数据采集 Sample Data (UC4)负责从各个传感器采集各种飞行数据；系统同步 Synchronize with SystemB (UC5) 负责实现与另一个冗余系统的同步工作，NAS 在设计上采用双机冗余的策略以保证其安全可靠；表决输入数据 Vote InputData (UC6)对收到的传感器采集数据进行验证和处理；表决输出数据 Vote OutputData (UC7)对计算后产生的飞行指令数据进行校验和处理；传输数据 Transmit Data (UC8)将各种表决后的数据传输给对方冗余系统；计算控制律 Calculate Control Law (UC9)，依据控制律计算公式使用表决处理后的飞行数据进行计算； 输出飞行指令(UC10)，将表决后的飞行参数发送给舵机作动器；关闭系统 Shut Down (UC11)，系统掉电后的一系列数据回写等操作。

第二个工业案例来自航海工业领域，它是一个典型的引擎控制系统(ECS)，该系统又分为几个功能模块，如阀门控制模块、燃油服务模块、航油运输模块等。在该实验中选用了其中的 27 个用况对其进行建模描述，例如，打开阀门、关闭阀门、控制阀门高度、维护阀门设置、启动燃油供给、启动循环泵、关闭循环泵、维护助推器设置、启动航油传输泵、维护航海传输设置等。

2.3.3 实验执行

RQ1 研究问题主要对 RUCM4RT 的实时需求建模能力进行评估，因此在该实验中分别使用 RUCM4RT 和 RUCM 对选取的工业案例进行用况建模，然后从各个实时特性指标对它们各自的建模能力进行比较。在该实验中设计如下的实时特性指标：周期任务(Periodic Task)、迸发任务(Burst Task)、不规范实时任务(Irregular Task)、非周期性实时任务(Sporadic Task)、并发任务(Concurrent Task)、时间点约束(Time Instant Constraint)、时间段约束(Time Duration Constraint)、存储约束(Storage Constraint)、数据精度(Data Precision)、实时事件(Timed Event)。实验中分别采用 RUCM4RT 和 RUCM 方法，对案例进行用况建模，并针对每个指标收集相应识别出的实例个数。

RQ2 研究问题主要对 RUCM4RT 生成 UML 定时图的有效性进行评估，在该实验中分别采用三种不同的用况生成策略：基于全条件覆盖(*All Condition Coverage*)标准、基于全事件流覆盖(*All FlowOfEvents Coverage*)标准、基于全语句覆盖(*All Sentence Coverage*)

标准，评估对 RUCM4RT 生成 UML 定时图的有效性。

2.3.4 实验结果

2.3.4.1 RQ1 的实验结果及分析

表 7 和表 9 针对实验中创建的 RUCM4RT 模型及其自动生成的 UML 定时图(Timing Diagram)的主要特征给出了描述性的统计分析。如表 7 所示，在实验中总共使用了 40 个用况，其中 27 个用况是实时用况。在所有的这些实时用况中，总共识别出并建模了 118 个实时性约束和 47 个其他的非功能需求约束(如，存储约束、数据精度、带宽等)。在实验中，识别并建模了 28 个参与者(Actor)，具体包含：9 个人 Actor、3 个系统 Actor、16 个资源 Actor。

为了进一步对 RUCM4RT 的建模能力进行分析，实验中分别针对每一个实时特性指标(2.3.3 节)对 RUCM4RT 用况建模与 RUCM 用况建模进行统计分析。在实验中，给定一个用况和实时特性指标，分别统计使用 RUCM4RT 识别出的建模元素的数量和使用 RUCM4 识别出的建模元素的数量，由于实验中总共选用了 40 个用况，这样就可以进行统计分析。

本实验中采用 Vargha-Delaney 统计[77]、Wilcoxon 秩和检验[79]对这些统计后的建模元素数量进行统计分析。其中，Vargha-Delaney 的统计计算数值是 $\hat{A}_{12}$，它是一个效应评价因子。在该实验中，给定一个实时特性指标，$\hat{A}_{12}$ 用来比较 RUCM4RT 和 RUCM 识别该实时特性元素的建模能力的高低。如果 $\hat{A}_{12}$=0.5，则 RUCM4RT 和 RUCM 具有同样的性

表 7 案例研究中 RUCM4RT 模型的描述性统计

Case Study	Actor			Regular UCs	Real Time Use Case (RTUC)				# of time-related constraints	# of other constraints
	Human Actor	External System	Resource Actor		# of Periodic RTUC	# of Burst RTUC	# of Irregular RTUC	# of Sporadic RTUC		
NAS	3	1	8	2	6	1	3	1	54	21
ECS	6	2	8	11	7	1	6	2	64	26
Sub Total	**9**	**3**	**16**	**13**	**13**	**2**	**9**	**3**	**118**	**47**
Total	**28**			**40**					**165**	

能；如果 $\hat{A}_{12}$>0.5，那么 RUCM4RT 有更高的机会发现更多的实时特性建模元素，反之 RUCM 有更高的机会发现更多的实时特性建模元素。Wilcoxon 秩和检验用来计算p-values，从而确定 RUCM4RT 和 RUCM 之间是否存在显著差异，该实验设定置信水平是 0.05，即如果一个p-value 小于 0.05 那么就存在显著性的差异。相应的统计数据在表 8 中进行报告。

表 8 Vargha-Delaney 统计、Wilcoxon 秩和检验(显著水平: 0.05)

实时特性指标	使用 RUCM4RT 建模 vs. 使用 RUCM 建模	
	$\hat{A}_{12}$	p-value
Periodic Task	0.7083	< 0.05
Burst Task	0.8624	< 0.05
Irregular Task	0.8531	< 0.05
Sporadic Task	0.8216	< 0.05
Concurrent Task	0.7946	< 0.05
Time Instant Constraint	0.9217	< 0.05
Time Duration Constraint	0.9108	< 0.05
Storage Constraint	0.9074	< 0.05
Data Precision	0.9308	< 0.05
Timed Event	0.9426	< 0.05

从表 8 中的统计数据可以发现，在识别周期任务(Periodic Task)时，RUCMRT 要比 RUCM 显著得好($\hat{A}_{12}$=0.7083 且p-value<0.05)。针对其他的实时特性指标，也可以发现 RUCMRT 要比 RUCM 显著得好。依据表 8 中的统计数据可以得出结论：RUCM4RT 针对实时系统的需求建模能力要比 RUCM 显著得好，即 RUCM4RT 更能有效的辅助需求人员对实时系统系统进行用况建模。

2.3.4.2 RQ2 的实验结果及分析

如表 9 所示，针对实时用况生成的用况场景，自动生成了相应的 UML 定时图(Timing Diagram)。具体地讲，基于全条件覆盖(*All Condition Coverage*)标准，自动生成了 418 个 UML 定时图；基于全事件流覆盖(*All FlowOfEvents Coverage*)标准，自动生成了 87 个

UML 定时图；基于全语句覆盖(*All Sentence Coverage*)标准，自动生成了 69 个 UML 定

表 9 实验中自动生成的 RUCM 定时图(Timing Diagram)的描述性统计

覆盖策略	UML 定时图中的建模元素	最小值	最大值	平均值	总数
All Condition	Lifeline	2	8	3	1254
	State invariant	7	67	21	8778
	Duration constraint	0	12	3	1254
	Instant constraint	0	4	2	926
	Time observation	0	8	4	1672
	Duration observation	0	12	3	1254
	自动生成的 UML 定时图的总量：418				
All FlowOfEvents	Lifeline	2	8	3	261
	State invariant	7	67	16	1392
	Duration constraint	0	10	2	174
	Instant constraint	0	4	2	174
	Time observation	0	4	2	174
	Duration observation	0	10	2	174
	自动生成的 UML 定时图的总量：87				
All Sentence	Lifeline	2	7	3	207
	State invariant	7	67	13	897
	Duration constraint	0	10	2	138
	Instant constraint	0	2	1	69
	Time observation	0	4	2	138
	Duration observation	0	9	1	69
	自动生成的 UML 定时图的总量：69				

平均用时 (单位：ms)		
All Condition	*All FlowOfEvents*	*All Sentence*
0.32	0.29	0.27

时图；它们相应的时间消耗分别是 0.32ms、0.29ms、0.27 ms。

此外，针对实验中自动生成的 UML 定时图，表 9 展示了具体的建模元素的相关信息，即给出了 UML 定时图中不同的建模元素在不同的用况中创建的数量。例如，当以全条件覆盖(*All Condition Coverage*)标准自动生成 UML 定时图时，在所有生成的 UML 定时图中，State Invairants 的平均数量是 21 个，State Invairants 的最少含有量是 7 个，State Invairants 的最多含有量是 67 个。

2.3.5 相关讨论

如表 7 所示，该实验采用 RUCM4RT 对两个工业案例中不同的实时用况进行识别和描述。其中识别并建模了 13 个周期性实时用况(periodic use case)、2 个迸发实时用况(burst use case)、9 个非规则实时用况(irregular use case)、3 个零星实时用况(sporadic use case)。这种对实时用况的划分方法是非常有用的，它以一种系统化的方式有效的帮助需求人员识别出不同的实时任务，如周期性实时任务、迸发性实时任务等；此外，由于 RUCM4RT 采用 UCMeta4RT 的形式化方式，强制需求人员在进行用况建模和需求描述时必须要系统、精确。例如，当创建一个周期性实时用况(periodic use case)，必须要指定相应的周期时间、用况本身的时间消耗，同时这些时间数值也都会被实例化为不同的元模型。

当使用 RUCM4RT 进行实时约束的规约描述时，可以发现这种结构化的设计和基于 UCMeta4RT 的形式化方式也是非常有用的，因为它提供一定程度的关注点分离的作用。所有的系统约束都与其相应的功能描述语句进行了分离，通过使用 RFS 关键字机制，它们被独立组织和管理。这样便于对不同的系统约束进行维护，而不是将它们直接嵌入到用况规约的不同事件流、不同需求描述语句中。本实验中，如表 7 所示，总共识别并建模了 118 个实时约束和 47 个其他非功能约束(如，通讯约束)，例如，图 5 所示，数据传输精度是 12.5 kb/s 的约束被建模为一个通讯约束，并将它从‘ALT_digital’中的第一个需求描述语句中分离出来，进行集中管理和维护。这种设计在工业实践中是非常有用的。例如，我们的航空工业合作伙伴，在进行 IMA 这种复杂实时系统的需求开发，除了要对复杂的功能需求进行捕获建模，还需要对各种实时约束、资源约束进行确切定义和准确描述，而且需求开发的过程往往要经过多次迭代，相应的需求变更也非常频繁；通过使用 RUCM4RT 这种设计，不仅可以系统化的帮助需求人员对各种不同的实时用况进行

识别建模，同时还大大提高需求变更维护的效率。

在本实验中，依据 RUCM 全条件覆盖(*All Condition Coverage*)标准，总共创建了 418 个 UML 定时图。依赖于 RUCM4RT 用况规约的规模(如分支数、需求描述语句的多少)，大量的 UML 定时图可能会被创建，然而，采用手工方式对用况规约及其自动生成的 UML 定时图进行需求评审可能会比较低效而且容易犯错。为了解决这个问题，本文的研究中采用一种基于搜索算法的选择优化方法，只选择那些相似度比较大的一组用况场景进行相应的需求评审，具体的实施方法在第 5 章中进行了详细的论述。

2.4 相关研究工作的对比

2.4.1 形式化验证方法

为了验证实时系统的时间特性，时序逻辑技术如 Metric Temporal Logic[84](MTL)、Timed Computational Tree Logic[85](TCTL)、Real-Time Graphical Interval Logic[86](RTGIL)被经常使用[87]。行为树(Behaviour Trees)[88]是一种基于时间自动机[89]的技术，它采用图形化标记符号的方式对系统的功能需求进行形式化规约验证。时间 Petri 网是另外一种形式的方法。文献[90]采用时间 Petri 网对异步并发统进行分析，文献[91]设计了一个通用的时间 Petri 网用来对系统的性能进行分析，文献[90]针对时间，定义其形式化语义，从而对各种时间 Petri 网模型进行评估。

以上这些技术都是形式化方法，在被有效的使用之前必须要对使用者进行系统化的培训。这些形式化方法通常都会设计一套相应的标记符号用于进行推理验证，所以当需求人员在使用它们时就不能像使用自然语言那样流畅，而是要求需求人员首先要将捕获的需求信息进行抽象加工，将其转化为对应的形式化标记符，再采用相应的归约公式进行描述。这些形式化方法设计的目的是对实时系统的某方面特性进行验证，而不是从需求捕获、需求描述的角度对需求人员提供支持，因此，它们应该被作为需求验证(Validation&Verification)的技术而不是需求建模、需求规约的技术。RUCM4RT 是从需求捕获和需求规约的角度出发，它采用一种限制性自然语言的描述方式对需求进行描述，同时采用 Meta-Model(元模型)的形式化方式对需求建模，因此 RUCM4RT 能够更好的帮助需求人员进行需求导出和需求规约。

2.4.2 基于 UML 的外廓

系统建模语言(System Modeling Language (SysML) [92])提出了需求图(Requirement Diagram)，这是一种半形式化的建模方式。SysML 中的需求图主要针对系统需求的导出、维护、一致性管理等方面给出指导和支持，针对需求规约描述仍然采用自由文本的形式。此外，SysML 的需求图是一种通用的技术，当采用它对实时系统进行需求规约是仍然需要进行额外的扩展，例如，如何有效的定义并描述实时约束、资源约束等。UML 外廓 SPT[93](UML Profile for Schedulability, Performance and Time (SPT))是对 UML 进行实时领域建模的第一个扩展，它定义了一些基本的时间概念以及进行系统调度和性能分析的概念。之后 SPT 被发展为全新的 UML 外廓 MARTE[71]并被替代, MARTE[71]针对嵌入式实时系统的建模和分析引入了全新的概念，特别是 UML 在实时领域中的工业实践为 MARTE 中的建模概念提供直接来源。MARTE 并未针对实时系统的需求开发给出任何指导，本文针对 MARTE 引入的实时建模概念进行系统化的分析和学习，将那些需要在需求阶段就必须要进行捕获和说明的建模要素进抽取，并细化为用况建模的要素，从而提出了 RUCM4RT 实时系统的用况建模方法。

UML 定时图(Timing Diagram)是在给定的一段时间内对系统各种行为的分析[94]。在系统开发的早期对系统时间属性进行分析依赖于系统的实时约束和时间观察值。本文提出的 RUCM4RT 可以针对实时用况生成的每一个用况场景，自动生成相应的 UML 定时图，这样可以方便需求人员、设计人员、开发人员对系统的时间特性进行一定程度的分析，从而保证产品开发的效率、质量、成本。

2.4.3 RUCM 扩展

最后，从 RUCM[8]扩展的角度分析，本文还与其他一些研究工作相关。Zhang[147]等基于 RUCM 方法提出了一种测试用例自动生成的方法，Yue[9]等提出一种测试用例建模方法 RTCM[9] (Restricted Test Case Modelling)可以自动生成可执行测试用例。文献[95]提出一种基于 RUCM 的测试用例生成方法。与这些基于 RUCM 的方法相比，本文提出的 RUCM4RT 方法可以看作是 RUCM 方法针对实时系统需求建模的一种领域应用。此外，RUCM4RT 还支持从 RUCM4RT 用况规约到 UML 定时图的自动转换。

2.5 本章小结

时间相关的系统属性(如时间约束)，是进行实时系统设计的重要考虑因素。在我们工业合作伙伴的工业实践中还缺少一套系统化的方法，基于自然语言的使用，对系统实时特性和其他系统约束(如资源约束)进行建模和规约，因此也无法在这个阶段开展相应的实时特性的验证。本文提出了一种系统化的用况建模方法(RUCM4RT)，针对实时系统的时间约束和其他系统约束进行需求建模和规约描述，采用 UCMeta4RT 元模型的机制对建模后的需求进行形式化。基于 RUCM4RT 用况规约描述，可以自动生成相应的 UML 定时图从而更好的辅助需求人员、设计人员和开发人员对系统的实时特性、资源约束在需求早期阶段开展相关的验证工作。最后选用两个工业案例对 RUCM4RT 的建模能力进行评估，实验结果显示 RUCM4RT 及其自动化工具可以有效的帮助需求人员进行实时系统的需求捕获和建模。

第3章 面向实时特性需求的用况建模方法

本章针对大型复杂实时系统的用况建模过程中的横切关注点问题，从用况模型的有效创建、模型的有效管理、模型的可复用角度出发，将面向方面的建模思想应用到用况建模中，从而支持横切关注点的分离和独立建模，最终实现面向实时特性需求的用况建模方法。3.1 节陈述了本章的研究问题，3.2 节论述了本章提出的面向实时特性需求的用况建模方法，3.3 节详细报告了实验设计及分析，3.4 节对相关的研究工作进行了对比，3.5 节总结了本章的研究内容。

3.1 研究问题的提出

复杂实时系统(如航空飞行控制系统、通讯控制系统、海底石油勘探系统)对用况建模有特殊的挑战。出于安全和可靠的考虑，这类实时系统在设计上通常都采用双机或多机的设计方案，即系统通常由功能类似甚至一样的多个对等子系统共同构成。例如，在案例 NAS 的开发过程，我们的工业合作伙伴采用双机设计策略。在这种系统的用况建模中，系统的某些特性(如通讯媒介、时间约束、资源约束)往往会横穿在不同的用况中，而这些特性往往又是决定系统的关键指标。如果直接对它们进行用况建模，则会造成用况图的混乱和用况规约片段的冗余，导致用况模型很难被理解和可复用。因此，需要使用一种建模技术能够将这些横切关注点(如实时约束)进行分离，对其进行独立建模和管理。

下面使用图 13 中的视频会议系统 VCS[96](Video Conferencing System)具体说明对复杂系统进行用况建模的横切关注点的问题。如图 13 所示，一个 VCS 系统负责向另外一些 VCS 系统(即端点系统)发送和接受多媒体流，在 VCS 系统中主要有音频和视频两种多媒体信息。一个 VCS 系统的主要功能有：创建音频会议、创建视频会议、切断音频会议、切断视频会议、开始会议演讲、停止会议演讲。其他的端点系统具有类似的功能。

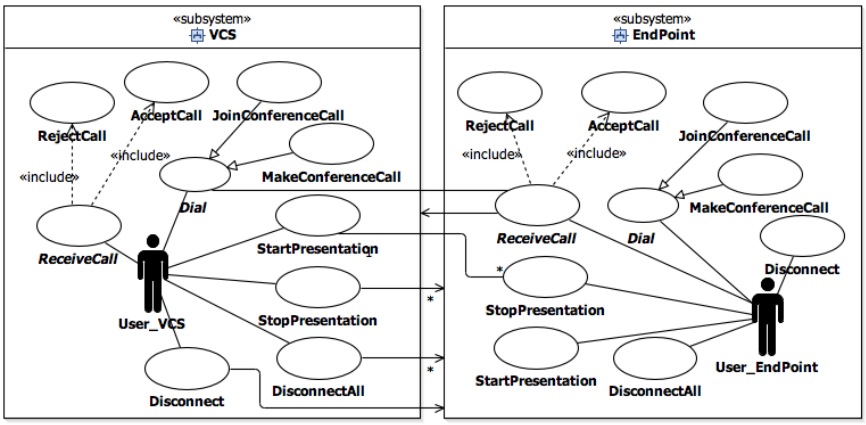

图 13 视频会系统 VCS 的用况图

图 13 中用两个 UML 包分别对 VCS 和端点系统的用况进行组织管理。由于这两个系统是通讯中的对等点，它们拥有相同的用况。然而，这两个系统的实现可能不一样，它们可能是具有相同功能的不同产品。图 13 中定义了用况之间的关联，例如，VCS 中的用况 *StartPresentation* 和 Endpoint 中的用况 *StopPresentation* 之间存在一个关联，并且该关联端点 StopPresentation 的多重性是(*)。这意味着，当 VCS 中的会议演讲启动时，系统必须要停止其他端点系统中正在进行的会议演讲。这种用况之间的关联关系在标准 UML 用况建模中是禁止的，此处的使用只是为了形象地说明系统的需求约束。

综上，如何解决复杂实时系统用况建模过程中的横切关注点分离的问题，是本章的研究问题。基于工业实践的需要，同时又受到面向方面需求工程[97](Aspect-Oriented Requirements Engineering (AORE))的启发，本文提出一种面向实时特性需求的用况建模方法 rtAsecptRUCM，它为 RUCM4RT(第 2 章)提供横切关注点分离的支持。RUCM4RT 和 rtAsecptRUCM 的搭配使用，实现对复杂实时系统的有效的用况建模，提供高质量可复用的用况模型。

3.2 面向实时性需求的用况建模方法

本节论述 rtAspectRUCM 方法：3.2.1 节论述了面向方面建模的概念模型；3.2.2 节论述了 rtAspectRUCM 外廓；3.2.3 节阐述了切面用况模型的定义；3.2.4 节论述了编织指导规范；3.2.5 论述了使用 rtAspectRUCM 进行建模的指导规则；3.2.6 节论述了模型编织。

3.2.1 rtAspectRUCM 的概念模型

面向方面的建模方法是在系统设计阶段通过横切关注点实现关注点与基模型的分离。如图 14 所示，方面/切面(Aspect)描述了与基模型分离的横切(crosscutting)结构，即切面模型；切面模型不能独立于基模型而存在，而是通过如下建模元素建立与基模型的关联：连接点(Joinpoint)可以被认为是基模型中的模型元素；切入点(Pointcut)依据选取规则，识别具有相似特征的一类连接点；切面模型中通常还要包含在基模型中新引入的与关注点相关的模型元素，而切面模型的导言(Introduction)用来描述此类模型元素；通知(Advice)则是针对选取的切入点实施的操作。

图 14 展示了 rtAspectRUCM 的概念模型。概念 Aspect(“方面”或“切面”)描述了一个横切关注点，在本文的语境下，一个横切关注点就是一组系统需求，它们横穿在其他的需求中用于描述系统的功能、性能。一个“连接点”(Jointpoint)是一个模型元素，它对应着一个“切入点”(Pointcut)。一个“切入点”，可能会使用相应的“通知”(Advice)，如用况图中的用况 Use Case、参与者 Actor，或者是用况规约中的前置条件、后置条件、事件流中的需求描述语句。

理论上讲，用况图中的任何模型元素和用况规约中的任何构成部分都可以是“连接点”。本文的设计中，定义了以下 5 种类型的“连接点”：参与者 Actor、用况 Use Case、前置条件、后置条件、事件流中的需求描述语句，基于相应工业案例的分析研究，它们足够使用。一个“切入点”(Pointcut)挑选一个或多个具有相似属性的“连接点”。一个模型元素(如，参与者 Actor)可以通过两种方式被引入到切面模型：它可以作为一个没有与切入点相关联的新元素而引入，也可以是通过其他模型元素关联到某个切入点。

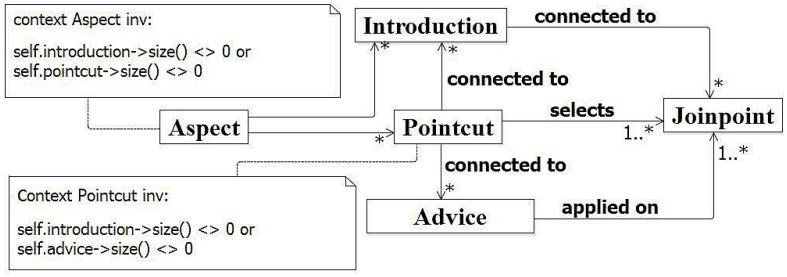

图 14 rtAspectRUCM 面向方面用况建模的概念模型

3.2.2 rtAspectRUCM 外廓

图 15 展示了 rtAspectRUCM 外廓。一个切面描述了一个横切关注点，被定义为衍型

«Aspect»，它继承自 UML 中的 Package。切面«Aspect»有两个属性：*baseUCM*，给出了一组基用况模型元素的名字并用逗号‘,’进行区别，针对其中的每个基用况模型元素，都会有一个切面模型被编织进该基用况模型。属性 *name* 是当前切面的名字。rtAsepctRUCM

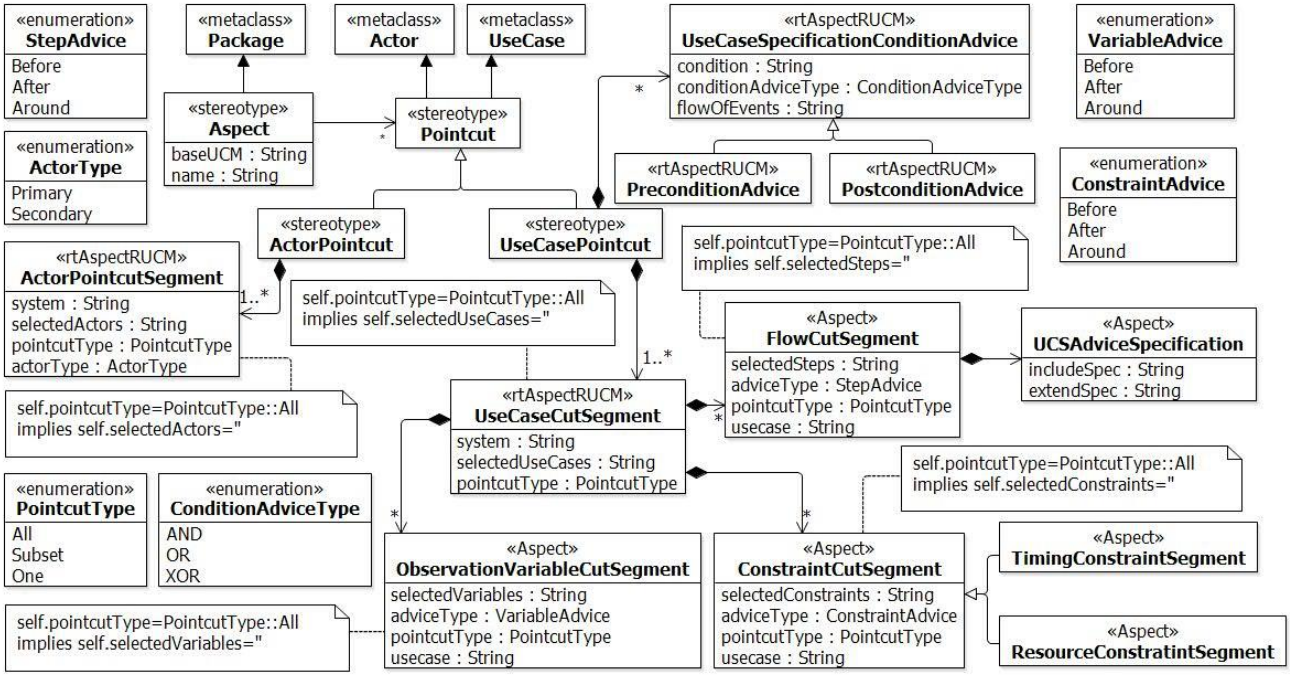

图 15 rtAspectRUCM 外廓

采用包对模型元素 Actor 和 UseCase 进行分组，以便对一个横切关注点进行定义。例如，图 16 所示，使用衍型«Aspect»进行标记的包 *NetworkDegradation* 包含用况 *AdaptCallRate*，参与者 *Timer* 等。 另一个例子，如图 17 所示，横切关注点 *Standby* 被建模为一个切面模型：一旦系统‘VCS ’ 空闲了 5 分钟，它的待机行为将会被激活；当系统‘VCS ’处于待机时，一旦系统‘VCS ’的任何参与者执行了任何行为活动，系统‘VCS ’都会被激活。使用包对一个切面模型中的元素进行管理的好处是：那些包含在未使用 rtAspectRUCM 衍型标记的包中的元素可以默认为是将要被引入到基用况模型中的新建元素。

如图 15 所示，一个使用切面的用况模型可能有一个或多个切入点。rtAspectRUCM 中定义了两种切入点(Pointcut)：用况切入点«UseCasePointcut»、参与者切入点«ActorPointcut»，这两种切入点是抽象衍型«Pointcut»分别针对用况 Use Case 和参与者 Actor 的不同实例化。

3.2.2.1 用况切入点(Use Case Pointcut)

一个用况切入点«UseCasePointcut»选取了一个或多个用况以及相应的用况规约中的

事件流(flow-of-events)，如图15所示，这可以通过衍型«UseCasePointcut»和类*UseCaseCutSegment*之间的组合关联(composition)关系实现，而类*UseCaseCutSegment*又与类*FlowCutSegment*进行关联。在类*UseCaseCutSegment*中，属性‘selectedUseCases’(selectedUseCases: String)指明了当前选取的用况，属性‘system’(system: String)指明了被选取用况的所属系统，属性‘pointcutType’(pointcutType: PointcutType)说明了当前选取的范围，即，全部(All)用况、部分(Subset)用况、一个(One)用况。如图16所示，用况*SelectedUseCases*使用了衍型«UseCasePointcut»，相应的衍型属性值表明：系统‘VCS’和系统‘EndPoint’中的所有用况都被选取；用况*AdaptCallRate*将会使用所有选取的用况进行延伸，并且它会被时间参与者*Timer*周期性的激活。图17中，衍型«UseCasePointcut»的属性值说明，切面模型*SelectedUseCases*通过延伸«extend»关联引入了另外两个用况：用况*Standby*、用况*ExitStandby*。用况*Standby*由时间参与者*Timer*触发，而系统‘VCS’和‘EndPoint’的任何参与者都可以激活用况*ExitStandby*，如图17中参与者切入点*SelectedActors*所示。

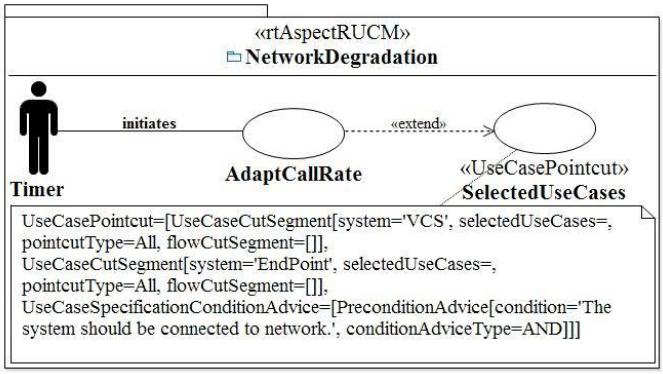

图 16 面向方面的用况图(*Network Degradation*)

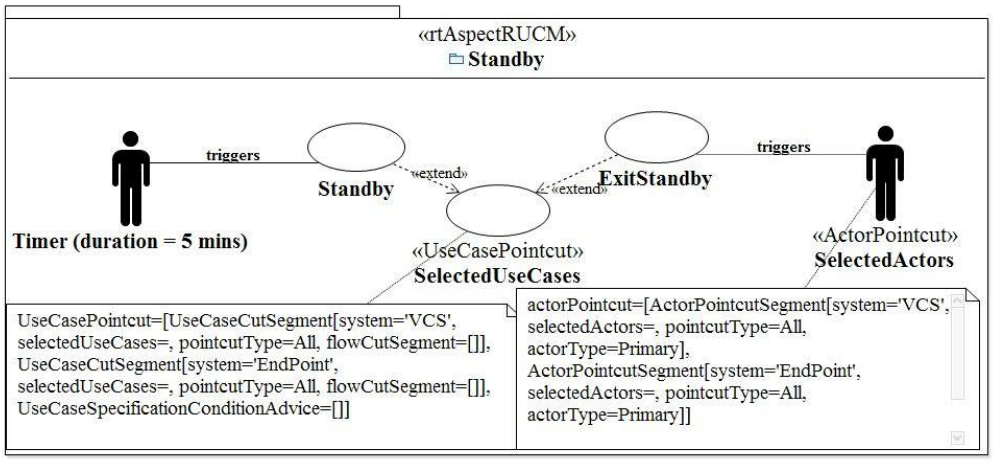

图 17 面向方面的用况图(*Standby*)

如图 15 所示，UseCasePointcut 至少要有一个 UseCaseCutSegment。一个 UseCaseCutSegment 由零到多个 FlowCutSegments 组成，其中 FlowCutSegments 描述了选取的事件流中的需求描述语句，同时必须使用“通知”指明相应的作用范围，即，之前、之后、替换。切入点的选取类型也必须使用“通知”进行说明，即，从一个用况规约中选择：全部、部分、一个需求描述语句。当不需要涉及用况规约的层次时，一个 UseCaseCutSegment 可能不会包含任何的 FlowCutSegment。当一个 UseCaseCutSegment 或者 FlowCutSegment 的切入点类型是 PointcutType::All 时，不需要显示的指明类 UseCaseCutSegment 的属性值 selectedUseCases，也不需要指明类 FlowCutSegment 的属性值 selectedSteps，该约束通过使用 OCL 进行了形式化描述(图 15)。例如，如图 17 所示，用况切入点 SelectedUseCases 由两个 UseCaseCutSegments 构成：一个用于从系统‘VCS’中选择所有的用况，另一个从系统‘EndPoint’中选择所有的用况。在这两个 UseCaseCutSegments 中，都不需要指定任何的 FlowCutSegment。

如果在切面模型中没有为一个新引入的用况指定相应的用况规约或者切面模型不需要涉及到用况规约的层次，那么这个用况将会通过三种不同的关联方式被编织到基用况图模型中：延伸 «Extend»、包含 «Include»、泛化 «Generalization»，其中这三种关联关系在切面模型的用况图中被明确的捕获。此外，用况切入点还需要指明选取的用况中的相应的语句(sentence)，这样新引入的用况可以进行延伸 «Extend» 或者被包含 «Include»。这可以通过 FlowCutSegment 和 UseCasePointcut 中的 UCSAdviceSpecification 实现。类 UCSAdviceSpecification 中的属性 includeSpec、extendSpec 分别指定了两个需求描述语句：*INCLUDE USE CASE <新引入的用况的名字>* 和 *EXTENDED BY USE CASE <新引入的用况的名字>*。在模型编织的过程中，这两个语句会在 FlowCutSegment 中选择的语句的前面、后面进行插入，或者替换原有的语句。针对新引入用况的延伸 «Extend» 或者包含 «Include» 的那些用况，相应的延伸或包含点已被新引入的用况进行了明确的描述，因此不需要在用况切入点描述模型中添加额外的信息。

针对实时用况规约中的事件观察变量 ObservatinVariable(图 7 所示)，设计 ObservationVariableCutSegment，这样可以将事件观察变量引入到选取的用况规约中。新引入的事件观察变量可以通过三种方式编织到 ObservationVariableCutSegment 中选择的事件变量中：将新引入的时间变量插入到 ObservationVariableCutSegment 中选择的事件

变量的前面、将新引入的时间变量插入到 ObservationVariableCutSegment 中选择的事件变量的后面、用新引入的时间变量替换 ObservationVariableCutSegment 中选择的事件变量。这三种编织方式被定义成枚举 ConditionAdviceType(如图 15 所示)。

针对实时用况规约中的约束(图 8 所示)，设计元类 ConstraintCutSegment，这样可以不同的约束引入到选取的用况规约中。ConstraintCutSegment 又实例化为：TimingCostraintSegment 和 ResourceConstraintSegment，分别对应实时约束和资源约束。新引入的约束可以通过三种方式编织到 ConstraintCutSegment 中选择的约束中：将新引入的约束插入到 ConstraintCutSegment 中选择的约束的前面、将新引入的约束插入到 ConstraintCutSegment 中选择的约束的后面、用新引入的约束替换 ConstraintCutSegment 中选择的事件变量。这三种编织方式被定义成枚举 ConstraintAdvice (如图 15 所示)。

针对用况规约中的前置条件和后置条件，设计 UseCaseSpecificationConditionAdvice 的两个特殊子类型：PreconditonAdvice、PostconditionAdvice，这样可以将用况的条件描述语句引入到选取的用况规约中。 新引入的描述语句可以通过三种方式编织到基用况规约模型中：联合、或者、排他，其中这种编织方式被定义成枚举 VariableAdvice(如图 15 所示). 如图 16 所示，用况切入点 SelectedUseCase 中有一个 PreconditionAdvice，其相应的条件描述是“The system should be connected to network”，这个用况前置条件应该被编织进用况切入点选取的用况的用况规约的前置条件描述中。编织的方式采用联合，这由 UseCaseSpecificationConditionAdvice 的属性 conditionAdviceType: ConditionAdviceType 的属性值：“AND”给出。

3.2.2.2 参与者切入点(Actor Pointcut)

一个参与者切入点选取一个或多个参与者，并由一个或多个 ActorPointcutSegments 构成。 ActorPointcutSegments 描述了选取的所有参与者、这些参与者所属的系统，以及切入点的类型。如图 17 所示，参与者 SelectedActors 使用了衍型«ActorPointcut»，相应的属性值显示：系统‘VCS’和‘EndPoint’中所有的参与者都被选取了。如同 UseCaseSegment 一样， 如果一个参与者切入点片段 ActorPointcutSegment 的类型 poincutType 是 PointcutType::All，那么就不需要指定选取的参与者 selectedActors。参与者有两种类型：Primary 和 Secondary，它们被定义为枚举 ActorType，这可以简化参与者切入点的表达式。例如，图 17 中参与者切入点选择了基用况模型中所有的首要参与

者。

3.2.3 切面用况模型的定义

一个切面用况模型 M 是一个使用了衍型«Aspect»的 UML 包，它可以包含以下的 UML 2.0 用况图元素:

1. 一组参与者，并且参与者的类型只能是如下之一：
   a) M中的一个参与者a可以是一个切入点pointcut，它负责从使用衍型«Aspect»的基本模型中选择一个、部分或全部的参与者。
   b) M中的一个参与者a可以是一个新的参与者，它是新被引入到基用况模型中的。在这种情况下没有使用任何的rtAspectRUCM衍型。
2. 一组用况，并且用况的类型只能是如下之一：
   a) M中的一个用况u可以是一个切入点pointcut，它负责从使用衍型«UseCasePointcut»的基本模型中选择一个、部分或全部的用况。
   b) M中的一个用况u可以是一个新的用况，它是新被引入到基用况模型中的。在这种情况下没有使用任何的rtAspectRUCM衍型。
3. 一组关系，其中每个关系只能是如下四种之一，且它们都是新被引入到基用况模型中的：
   a) M中的一个关系r可以是一个包含«include»，用于关联切面用况模型中的两个用况。
   b) M中的一个关系r可以是一个延伸«extend»，用于关联切面用况模型中的两个用况。
   c) M中的一个关系r可以是一个泛化，用于关联切面用况模型中的两个用况。
   d) M中的一个关系r可以是一个关联，用于关联切面用况模型中的两个用况。

3.2.4 定义编织指导规范

每一个横切关注点被建模为一个独立的切面用况模型。需要采用一种明确的顺序方式将各个横切关注点对应的切面模型编织到它们相应的基用况模型中，以确保编织后的用况模型的正确性。为了达到这个目标，模型编织顺序必须被定义并作为一个输入提供给模型编织器。然而，UML 用况图并不具备这样的能力，因此，rtAspectRUCM 采用 UML 的交互概览图对这些顺序进行描述，并把这种编织顺序定义为编织指导规范。图 18 给出了一个实例。

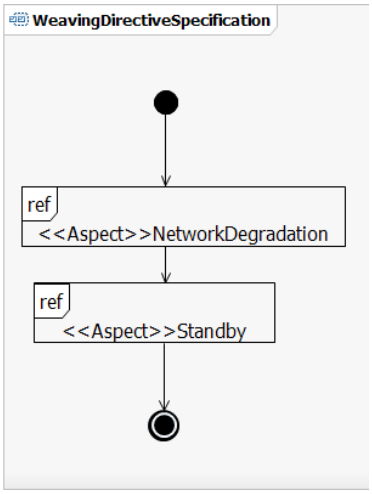

图 18 编织指导交互概览图的一个实例

UML 交互概览图通过使用 UML 活动图的一种变体对各种交互进行定义，其中这种 UML 活动图的变体主要是提供控制流的概要描述[98]。rtAspectRUCM 的编织指导规范定义了所有切面模型之间的交互，并借助 UML 活动图中的流控制特性(如判断(decision)、汇合(join)、分岔(fork))进行定序。当然也可以使用 UML 活动图进行相应的设计，选择 UML 交互概览图是因为它更简洁，即 UML 交互概览图中不出现消息 Messages 和生命线 Lifelines。

一个 rtAspectRUCM 编织指导规范包含以下模型元素：

1)一个起点(initial node)；

2)一组交互使用，其中每一个交互使用都对应着一个切面用况模型；

3)一组控制流的边，每一个控制流的边只能是以下两种类型中的一种：

a) 由起始点指向一个交互使用的控制流，其中的交互使用表示第一个编织的切面模型；

b) 一组用于连接交互使用(如判断(decision)、汇合(join)、分岔(fork))的控制流，用于表明相应的切面模型被编织到基用况模型中的顺序。

4)一个终点(final node)。

3.2.5 建模指导规则

rtAspectRUCM 外廓(3.2.2 节)提供了一套标记符，用于将横切关注点建模为相应的切面用况模型。在使用 rtAspectRUCM 之前，如图 19 所示，首先需要在需求层次对横切关注点进行识别(A1)。可以采用不同的方法(如，[99])实现这个目标，rtAspectRUCM 方法可以与这些现存的技术进行搭配使用。

系统的主要需求被建模为 RUCM4RT 模型(A2)，这样就创建了基用况模型 Base UCMod。 紧随其后的建模活动 A3，使用 rtAspectRUCM 对横切关注点进行建模描述。该活动又包含以下的子活动：使用衍型«Aspect»创建一个 UML 包；然后创建并描述切入点 pointcut(s)并创建相应的用况图中的其他模型元素(如，参与者 actors、用况 use cases)；最后，使用 RUCM4RT 需求描述模板对新引入到切面用况模型中的用况规约进行描述。这一活动的输出制品是一组切面用况模型，它们对应于识别出的每一个横切关注点。活动 A4，描述了编织顺序并给出了编织指导规范的交互概览图。

来自活动 A3 的切面用况模型 Aspect UCMods 将会被编织到 A2 中定义的相应的基用况模型 Base UCMod 中。依据 A4 中的编织指导规范的交互概览图给出的编织顺序，

A5 将会自动创建一个编织用况模型。A5 将会支持自动化的分析(A6)，例如，需求分析、UML 分析模型的自动转换、测试用例的自动生成。依据编织后的用况模型而不是那些独立的切面模型和基本模型，有时能够更有效的进行不同的需求分析，例如，识别和管理不同关注点之间的需求冲突、折中平衡[100]。基于文献[8]，可以实现从编织后的用况模型到不同 UML 图的自动转换。如果后续制品(如，测试用例)的生成依赖于一个从需求层次的面向方面建模方法 (如 rtAspectRUCM)到另外一种在设计层次或测试层次的面向方面建模方法 (如 AspectSM[96]) 的转换，就不需要在需求层次进行模型编织，相应的

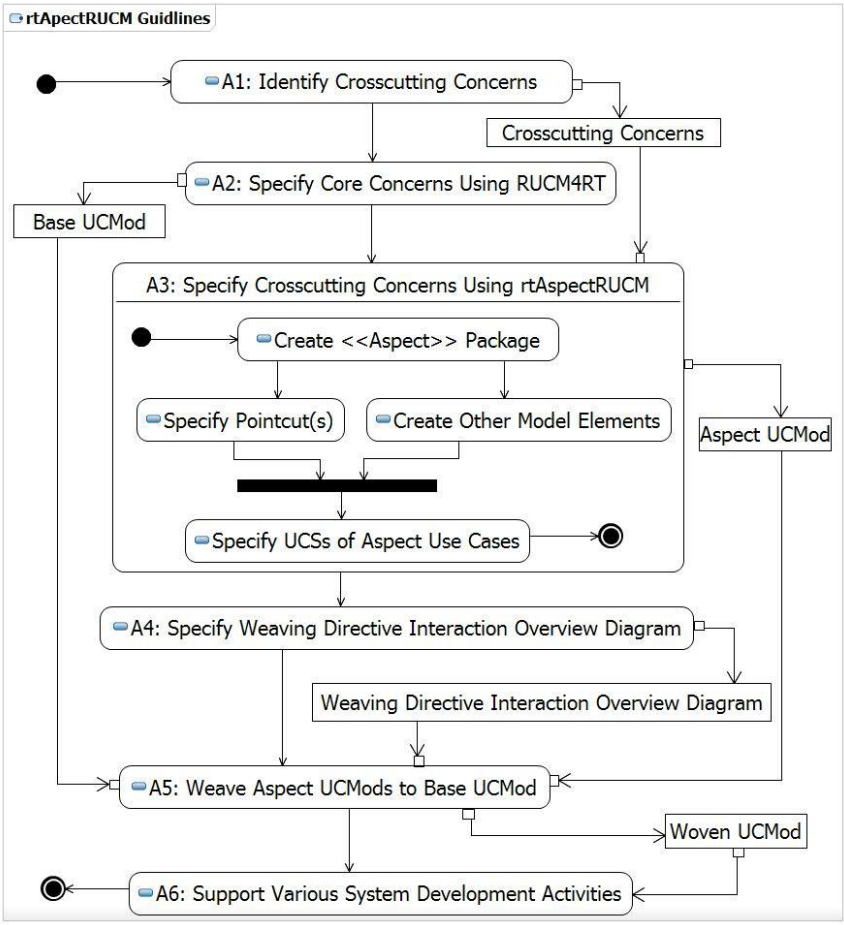

图 19 使用 rtAspectRUCM 建模的指导规则

活动 A4、A5 、A6 也就不需要了。

3.2.6 模型编织

rtAspectRUCM 的设计目标是将关注点分离的思想应用于用况建模中，即，实现横切关注点(如，实时约束、资源约束(图 8))与不同的 RUCM4RT 用况(如，周期性实时用况 PeriodicRTUC 图 7)的分离。通过引入关注点分离的思想，一方面可以支持需求人员比较

高效率的使用 RUCM4RT 进行用况建模；另一方支持对横切关注点的独立管理，特别是当需求开发过程需要进行多次迭代和频繁修改时。

系统的那些主要考虑被表述为系统的主要功能，可以使用 RUCM4RT 将其建模为不同的用况，即基用况模型(Base Use Case Model)。横切关注点可以是贯穿在不同用况中的功能需求和非功能需求，这些横切关注点可以使用 rtAspectRUCM 建模为不同的切面用况模型(3.2.3 节)。

模型编织主要负责将切面用况模型编织到基用况模型中，它需要读取切面用况模型、基用况模型、编织指导交互概览图(即模型编织的顺序)，并产生一个编织后的用况模型。由于 RUCM4RT 和 rtAspectRUCM 都是采用基于 UCMeta 的形式化方式，因此切面用况模型和基用况模型中的文本需求描述都会被实例化为不同的 UCMeta 实例，其中切面用况模型被形式化为 rtAspectRUCM 的扩展 UCMeta 实例，而基用况模型被形式化为 UCMeta4RT 的实例。这种基于 UCMeta 元模型的形式化方式为模型编织提供直接支持。例如，*ActorPointCut* 和 *UseCasePointCut* 定义了如何将横切关注点(如，参与者、用况规约片段、约束、事件观察变量)编织到相应的基用况模型中，以及在基用况模型中的具体的编织位置(如 *StepAdvice* 的取值指明了是在选择语句的之前(Before)、之后(After)进行织入，还是替换(Around)原有的语句)。图 20 展示了相应的模型编织算法。

WeaveUseCaseModel (b : UseCaseModel, a : UseCaseModel, w : UseCaseModel)

Inputs:

b: A formalized base UCMod, which is an instance of UCMeta4RT

a: A formalized aspect UCMod, which is an instance of UCMeta extended with the rtAspectRUCM profile

Output: *w*: A woven UCMod, which is an instance of UCMeta4RT

Algorithm:

1. 1. For each use case *uc* stereotyped with «UseCasePointcut» in *b*, do
2. A. Query the base UCMod *b*, based on the values of the attributes of «UseCasePointcut» and obtain a collection of use cases *SelectedUCs*.
3. B. Query the aspect UCMod a, obtain a collection of model elements meles (either use cases or actors) that are not stereotyped with «UseCasePoincut» or «ActorPointcut» but are directly or indirectly connected to use case uc.
4. C. Add meles to the woven UCMod and link them to each of SelectedUCs through the associations specified in the aspect UCMod a.
5. D. If uc contains an instance of UseCaseSpecificationConditionAdvice, then introduce PreconditionAdvice and/or PostconditonAdvice to each of SelectedUCs, according to the specification of the advices.
6. E. For each selected use case suc of SelectedUCs, do
7. a. For each use case *auc* of *meles*, do

8. i. If *auc* extends *suc* as specified in the aspect UCMod *a*, update the UCS of *suc* by
9. a) Adding *auc* to the field of 'Extending Use Case', and
10. b) Adding a *sentence* EXTENDED BY USE CASE <name of the *auc*> Before, After, or Around the selected steps specified in FlowCutSegment contained in «UseCasePointcut».
11. ii. If suc extends auc, update the UCS of auc by
12. a) Adding *suc* to the field of 'Extending Use Case', and
13. b) Adding a sentence EXTENDED BY USE CASE <name of the *suc*> Before, After, or Around the selected steps specified in FlowCutSegment contained in «UseCasePointcut».
14. iii. If auc includes suc, update the UCS of auc by
15. a) Adding *suc* to the field of 'Included Use Case' and
16. b) Adding a sentence INCLUDE USE CASE <name of the *auc*> Before, After, or Around the selected steps specified in FlowCutSegment contained in «UseCasePointcut».
17. iv. If suc includes auc, update the UCS of suc by
18. a) Adding *auc* to the field of 'Included Use Case' and
19. b) Adding a sentence INCLUDE USE CASE <name of the *suc*> Before, After, or Around the selected steps specified in FlowCutSegment contained in «UseCasePointcut».
20. v. If auc contains an instance of ObservationVariableCutSegment ovs, then update the UCS of suc by
21. a) Introducing a new instance of ObservationVariable *ov* and
22. b) Adding *ov* Before, After, or Around the selected ObservationVariables specified in ObservationVariableCutSegment *ovs*.
23. vi. If auc contains an instance of ConstraintCutSegment cons, then update the UCS of suc by
24. a) Introducing a new instance of TimingConstraintSegment *rtcons* or a new instance of ResourceConstraintSegment *rescons*, according to the specification of ConstraintCutSegment *cons*
25. b) Adding *rtcons* or *rescons* Before, After, or Around the selected costraints specified in ConstraintCutSegment *cons*.
26. vii. If auc specializes suc, update the UCS of suc by adding auc to the field of 'Specialized Use Case'.
27. viii. If suc specializes auc, update the UCS of auc by adding suc to the field of 'Specialized Use Case'.
28. 2. For each actor acr stereotyped with «ActorPointcut» in b, do
29. A. Query the base UCMod *b*, based on the values of the attributes of «ActorPointcut» and obtain a collection of actors *SelectedActors*.
30. B. Connect each selected actor to the newly-added use cases to the woven UCMod according to what is specified in the aspect use case diagram.
31. C. For each actor of SelectedActors, do
32. a) If the actor is a primary actor, according to the information contained in the ActorPointcutSegment of «ActorPointcut», add this actor to the field of 'Primary Actor' of the UCSs of the use cases that are connected to the *actor* as specified in the aspect use case diagram.

33.	b)	If the actor is a secondary actor, according to the information contained in the ActorPointcutSegment of «ActorPointcut», add this actor to the field of 'Secondary Actor' of the UCSs of the use cases that are connected to the actor as specified in the aspect use case diagram.

图 20 模型编织算法

3.3 案例研究

本节通过三个工业案例对提出的 rtAspectRUCM 进行评估。

3.3.1 案例描述

该实验选用了来自不同领域的三个工业案例，它们分别是：通讯领域的系统 VCS[96]、海底石油系统 SOPS[101](Subsea Oil Production System)和航空领域的飞行控制系统(NAS)。表 10 描述了这些系统对应的基用况模型和切面用况模型的相关特性。

表 10 基用况模型和切面用况模型的特征描述

System	# of Base Use Cases	Total # of UCSs	# of Aspect UCMods	# of Actors
VCS	40	10	8	5
SOPS	65	12	6	9
NAS	46	11	7	9

VCS[96]。VCS 是一个会议系统并由四个对等的端点系统构成，这些端点系统具有相同的功能。其中每个端点系统包含 10 个用况，这样 VCS 中总共包含 40 个用况。 VCS 的主要功能是对多媒体流的发送、接收进行管理。在 VCS 中，音频信号、视频信号使用独立的信道进行传输。在任何时刻，会议演讲只能由一个会议参加者实施，而其他会议参加者只能接收当前的会议演讲。VCS 的每一个端点系统都有一个相应的人参与者(Human Actor 图 6)进行使用，一个定时器 Timer 会周期性的触发"呼叫"。 依据[96]，本实验使用 rtAspectRUCM 定义了以下 8 个横切行为： *Network Degradation*、*Standby*、*Media Quality Recovery*、*Do Not Disturb*、*Synchronization Mismatch*、*Intelligent Packet Loss Recovery*、*Echo Reduction*、*Noise Cancellation*。

1. *Network Degradation*：一旦通讯媒介发生故障并导致通讯中出现坏包、重复包，这个横切行为就会被激活。
2. *Standby*： 当VCS空闲了特定时间后，它就会采取Standby行为。当VCS处于Standy模式，对VCS进行任何的活动都会使VCS处于活跃模式。

3. *Media Quality Recovery*：VCS的一个稳健性行为就是对媒体(音频、视频)的品质损失进行恢复。在一个视频会议中，VCS都会每隔一定时间对视频、音频信号的质量进行检测。如果媒体信号的质量处于有效范围内，VCS就会继续正常操作，否则它就会尝试对音频、视频信号进行品质恢复。如果能够恢复成功，它将继续正常操作，否则VCS将会重启。
4. *Do Not Disturb*：一旦VCS处于*Do Not Disturb*，它将会忽略所有的call in。如果VCS已经在一个会话中，它将会保持原有的会话并忽略新来的呼叫。
5. *Synchronization Mismatch*：一旦音频信号和视频信号之间不同步，特定算法就会被使用来减少视频和视频之间的不同步。
6. *Intelligent Packet Loss Recovery*：一旦视频会议中发生网络丢包，特定的算法将会被使用来处理数据丢包。
7. *Echo Reduction*：一旦在视频会议或语音会话中出现回音，VCS将会启动回音消除算法进行相应处理。
8. *Noise Cancellation*：在视频会议中，VCS可能会使用特定的算法进行噪音消除。

SOPS[101]。SOPS 是一个系统之系统(systems of systems)，主要对石油、天然气的生产进行管理。 SOPS 有四种不同类型的系统，其中三种系统部署在海平面上，另外一种系统部署在海洋底。这些系统有着不同的功能，它们使用不同类型的通讯媒介进行通讯。该实验中从 65 个代表性的用况中挑选出 12 个进行建模，并使用 rtAspectRUCM 定义了六个横切行为：*Operation Mode Exchange*、*Backup Communication*、*Communication Timeout*、*Runtime Configuration*、*Communication Bandwidth Limiting*、*Data Update Mechanism Switch*。

1. *Operation Mode Exchange*：系统的操作模式可以在正常模式和仿真模式之间进行切换。在每种模式下，系统可能会有重叠的功能。
2. *Backup Communication*：对应SOPS这类安全关键系统，当安装了光导纤维，需要创建一个海洋外面和海底之间的电路通讯系统作为备用通讯系统。 相应的，所有的功能在备用通讯模式下也必须可用。然而，在有限带宽的条件下，某些功能可能在备用通讯模式下不能使用，它们需要被禁用。
3. *Communication Timeout*：顶端控制系统等待海底控制系统的反馈。如果等待超时，顶端控制系统发送给海底控制系统的请求就会从消息等待队列中被移除。

4. *Runtime Configuration*：当系统进行配置运行时，需要对其中部署的控制软件进行参数配置和调试。
5. *Communication Bandwidth Limiting*：当带宽有限并且控制系统需要连接到外部以太网设备时，带宽限制将会非常重要。通讯基础设施的带宽以及是否使用速度较慢的备用通讯线决定了带宽限制的具体要求。这样的通讯带宽限制将会直接影响系统的不同功能。
6. *Data Update Mechanism Switch*：通常采用两种机制将海底系统发送的数据传输给顶端控制系统：依据顶端系统和海底系统之间的通讯联络的拥堵情况，动态的将一种数据切换为另一种数据；使用顶端控制设备中的操作器进行手工切换。这两种机制之间的切换会对不同的系统功能产生影响。

NAS[102]。NAS 是一个飞行控制系统，2.3.2 节对 NAS 的某些用况进行了详细描述。NAS 采用双机容错的设计，即它由两个对等的控制系统组成。在该实验中针对每个对等系统选了 23 个用况进行建模，这样 NAS 总共有 46 个用况进行实验。针对 NAS 使用 rtAspectRUCM 定义如下七种不同的横切行为：*System Synchronization*、*Flight Mode Exchange*、*Periodical Action*、*Data Monitoring*、*Data Voting*、*Fault Handling*、*Communication Timeout*。

1. *System Synchronization*：NAS采用双机设计，因此在每个飞行周期开始时都要进行系统的同步，而这些同步操作在对等系统中是一样的。
2. *Flight Mode Exchange*：NAS有两种飞行模式：自动驾驶和人工驾驶。飞行员在飞行过程中可以进行飞行模式的切换，不同的驾驶模式下会有相同的系统功能。
3. *Periodical Action*：在每个时钟周期，系统都会执行不同的飞行任务，如采集飞行数据、计算飞行指令。
4. *Data Monitoring*：数据监控是飞控系统的一种重要容错手段，它用于保证数据的有效性和安全性。在双系统的设计中，每个对等系统都会对自己采集和对方传送的数据进行有效性识别。
5. *Data Voting*：数据表决是另一种重要的容错手段，它主要用于选取最优的数据进行计算。在双系统的设计中，每个对等系统都会对自己采集和对方传送的数据进行择优选择。

6. *Fault Handling*：主要在周期任务中对各种故障进行紧急处理。
7. *Communication Timeout*：飞行控制系统需要跟其他外部系统(如，航电系统)进行通讯。如果等待超时，飞行控制系统发送给外部系统的请求就会从消息等待队列中被移除。

3.3.2 实验评价指标

实验目标：对 rtAspectRUCM 的建模工作量进行评估。

实验评价指标：rtAspectRUCM 的设计目标就是要提高用况建模的效率，即通过关注点分离的思想实现用况建模自身过程中的可复用性。因此，在该实验中通过计算使用 rtAspectRUCM 进行用况建模时的建模工作量对 rtAspectRUCM 进行评估。该实验中的建模工作量(即实验评价指标)定义为：给定一个实验对象(工业案例：VCS、SOPS、NAS)，使用不同的用况建模方法(即 rtAspectRUCM、RUCM4RT)需要创建的用况模型元素(如用况、参与者、关系)的数量。然后针对横切关注点的用况建模，可以使用这个评价指标对 rtAspectRUCM 和 RUCM4RT 进行比较。

3.3.3 实验结果及分析

针对每个工业案例(VCS、SOPS、NAS)，分别使用 rtAspectRUCM 和 RUCM4RT 进行用况建模。针对每个工业案例中的每个横切关注点进行相应的统计。当使用 rtAspectRUCM 时，统计如下建模元素的数量：用况、参与者、关系(关联、延伸、包含、泛化)、切入点(pointcut)；当使用 RUCM4RT 时，需要统计如下建模元素的数量：用况、参与者、关系(关联、延伸、包含、泛化)。表 11 报告了所有的建模任务及其相应的实验结果，该实验并没有考虑创建相应的用况规约的建模成本，即该实验仅在构建用况图的层次进行建模成本的比较。

VCS 案例分析。VCS 中总共有 8 个横切关注点，3.3.1 节对其进行了详细描述。从表 11 中的实验数据可以发现：使用 rtAspectRUCM 对这 8 个横切关注点进行建模能够显著性的减少用况之间关系(延伸、包含、泛化)的建模成本，即平均可以节省 95% (=(440-20)/440)的成本(表 11 所示)。具体讲，针对 VCS 中的 8 个横切关注点，使用 rtAspectRUCM 需要创建 20 个关系模型，而使用 RUCM4RT 则需要创建 440 个关系模型。从参与者 Actor 的角度进行比较，使用 rtAspectRUCM 需要创建 10 个参与者，而使用 RUCM4RT

需要创建 8 个参与者。从用况角度进行比较，使用 rtAspectRUCM 需要创建 19 个用况，而使用 RUCM4RT 需要创建 11 个用况。在用况图的构建中，创建一个用况、一个参与者或者是一个关系的建模工作量，基本上可以认为它们是一样的。针对所有的 8 个横切关注点，使用 RUCM4RT 需要创建 459 个建模元素而使用 rtAspectRUCM 仅需要创建 58 个建模元素。 这意味着在该案例分析中，使用 rtAspectRUCM 平均节省了 87%((459-58)/459) 的建模工作量。

使用 rtAspectRUCM 同时还需要创建切入点，如表 11 所示，针对 VCS 总共创建了 10 个切入点，这些切入点创建是使用 rtAspectRUCM 的额外工作量。该案例中通过创建 10 个切入点可以节省 410 个模型元素的创建。如图 16-图 17 所示，创建一个切入点的建模工作量主要在于描述选取的用况、参与者以及它们对应的系统。因此，创建 10 个切入点的工作量要比创建 410 个模型元素的工作量小，故而，使用 rtAspectRUCM 可以减少单独使用 RUCM4RT 的建模工作量。 针对案例 SOPS，总共有 6 个横切关注的，如表 11 所示，使用 rtAspectRUCM 节省了 80%的建模成本。针对案例 NAS，总共有 7 个横切关注的，如表 11 所示，使用 rtAspectRUCM 节省了 82%的建模工作量。

表 11 三个工业案例的实验结果

Case Study	Crosscutting concerns	Using AspectRUCM					Without AspectRUCM				Effort saved (%)
		UCs	Actors	Rels	Pointcut	Total	UCs	Actors	Rels	Total	
VCS	1	2	1	2	1	6	1	1	40	42	86%
	2	3	2	4	2	11	2	1	80	83	87%
	3	3	1	2	1	7	2	1	80	83	92%
	4	3	2	4	1	10	2	1	80	83	88%
	5	2	1	2	1	6	1	1	40	42	86%
	6	2	1	2	1	6	1	1	40	42	86%
	7	2	1	2	1	6	1	1	40	42	86%
	8	2	1	2	1	6	1	1	40	42	86%
	Total	19	10	**20**	9	**58**	11	8	**440**	**459**	**87%**
SOPS	1	3	1	4	1	9	2	1	78	81	89%
	2	3	2	6	2	13	2	1	56	59	78%
	3	2	1	2	1	6	1	1	48	50	88%
	4	2	1	2	1	6	1	1	12	14	57%
	5	3	2	6	2	13	2	1	47	50	74%
	6	3	1	4	1	9	2	0	25	27	67%
	Total	16	8	24	8	56	10	5	266	281	**80%**
NAS	1	2	2	4	1	9	1	2	42	45	80%
	2	2	1	2	1	6	1	1	14	16	62%
	3	5	3	6	2	16	3	3	74	80	80%
	4	2	3	2	1	8	1	2	48	51	84%
	5	2	3	2	1	8	1	2	50	53	85%
	6	3	1	2	1	7	2	1	46	49	86%
	7	3	3	4	2	12	2	3	62	67	82%
	Total	19	16	22	9	66	11	14	336	361	**82%**

本实验中三个工业案例的实验数据显示：针对横切关注点，使用 rtAspectRUCM 可以显著性的减少建模工作量。在使用 rtAspectRUCM 时，需要分别使用衍型«UseCasePointcut»和«ActorUseCasePoint»创建用况切入点和参与者切入点。实验中的数据显示，针对一个用况模型，额外创建 80%的关系模型要比创建几个切入点更加耗时。

虽然实验数据显示了 rtAspectRUCM 的有效性，但是还需要进行受控实验(controlled experiments)进一步的确认，这需要涉及不同的人力、更多的案例、系统的实验设计，以计算 rtAspectRUCM 具体的建模成本节省。这是未来工作中的一部分内容，即对 rtAspectRUCM 进行实证研究(Empirical Study)，分析它在增强关注点分离、提供维护性、可复用性方面的性能，以及模型完整性、准确性的评估分析。

3.4 相关研究工作的对比

3.4.1 用况模板

在工业实践中通常采用一个模板对用况规约进行组织和描述，不同的用况模板(如[20]) 被设计来解决这个问题。这些用况模板都有一些通用的内容：用况名字、用况基本描述、前置条件、后置条件、基本流、备选流。文献调研[103]系统化的总结了将文本描述的需求转化成分析模型中的 6 种技术(如[104])，它们都需要使用用况，RUCM 就是基于这些最新研究创建的。

3.4.2 面向方面的用况建模

文献[105] 中提出一种面向方面的用况建模方法，它采用衍型«Aspect»化的关联关系实现从属用况与基用况的关联。基于用况中事件流的需求描述语句的通配符，定义了一个用于描述切入点表达式的语法规则。文献中给出了四种不同的编织“通知”：之前(before)、之后(after)、替换(around)、并列(concurrent)。该方法并没有针对用况图直接引入切面，因此既没有重用相应的图形化标记符也没有新引入。切面用况和相应的基用况被编织成一个 petri 网模型，从而为后续分析提供相关输入。rtAspectRUCM 对标准 UML 用况图进行扩展，通过使用 UML 衍型对 UML 原有的图形标记符进行扩展和重用。

文献[106]提出了一种面向方面的用况建模方法。该方法中，将延伸点的语义扩展为汇合点。 通过这种扩展，基用况模型需要被修改，即，将延伸点的文本语句直接插入到基用况模型对应的切入点用况的用况规约中。如果存在多个切入点用况(这在 AOM 环境下是非常有可能)，就需要对这些切入点用况的用况规约的多个地方进行修改。这意味着该方法没有真正的实现切面与基模型进行分离。此外，该方法只使用一种编织“通知”：切面用况中描述的延伸行为。

Sillito[107]等提出了一个文本的切面语言 AspectU，用以实现对用况模型中的横切关注点进行模块化。AspectU 切面可以转换成 AspectJ 代码实现。AspectU 是纯文本的，它跟编程语言非常类似，而 rtAspectRUCM 主要依赖于 UML 用况图中的图形化标记符。因此，从可用性角度说，rtAspectRUCM 应该要易于被工程师(特别是需求工程师)理解和使用。Mussbacher[108]等人借助用况映射图(use case maps)提出了一种面向方面的需求建模方法。 在该方法中，编织“通知”(advice)和切入点(pointcut)通过使用用况映射图(use case maps)中的图形化标记符进行定义。

此在，在文献调研中还发现有些研究工作(如[109])将切面的概念应用到基于目标的模型(goal models)中。其他一些面向方面的建模技术(如[96])被应用到软件开发生命周期的不同抽象层次而不是用况模型中。

3.5 本章小结

用况建模作为一种捕获、描述系统功能需求的技术已在工业实践中被广泛采用[110,6]。用况规约(Use Case Sepcification)是用况建模的重要制品之一，它通常是基于文本描述的，因此会不可避免的引入模糊性。为了尽可能地避免需求的模糊性，RUCM[23]用况建模方法被提出并通过实证研究(Empirical Study)证明其有效性和易用性。面对大型复杂实时系统(如，IMA 航空系统)的开发，大量的精力和工作投入到用况建模中(如，用况的规约描述、用况的维护)，因为这类系统的开发过程中，需求的质量对于每一个后续活动都会有重要影响。此外，大量的横切关注点散落在不同的用况中，这些横切关注点需要采用一种有效的方式进行管理并支持其可复用性。

本章提出了一种 RUCM 的扩展，即 rtAspectRUCM，它在用况模型层次实现了横切关注点的建模，从而可以辅助 RUCM4RT 对大型复杂的实时系统进行需求建模。rtAspectRUCM 采用了一个 UML 外廓将横切关注点建模为用况图和用况规约中的切面。rtAspectRUCM 扩展对于复杂实时系统的用况建模是必要的，在这类系统的用况建模中需要对额外的非功能属性进行捕获和描述。为了对 rtAspectRUCM 的有效性(即节省建模成本)进行评估，本文选用了三个工业案例进行实验分析。实验结果显示，针对横切关注点 rtAspectRUCM 平均可以节省 80%的建模成本。

第4章 基于工业标准的用况变异分析方法

本章主要从需求评审的支撑角度出发，解决用况评审中存在的两类典型问题：1) 缺少系统而精确的用况缺陷分类，2) 需要一种系统而有效的用况缺陷注入方法。为了对不同的用况评审技术(如基于检查列表的技术、基于视角的阅读技术)进行客观公正、有效可靠的评估，同时又可以支持评审实验过程的可再现，本章借鉴变异分析(Mutation Analysis/Mutation Testing)技术，将其应用到用况评审中，提出一种基于工业标准的用况变异分析方法。4.1 节阐述了本章的研究问题，4.2 节系统化论述了本章提出的用况变异分析方法，4.3 节对案例分析进行报告，4.4 节讨论相应的有效性风险，4.5 节对相关的研究进行对比，4.6 节总结本章内容。

本章提出的用况变异分析方法不仅适用于标准的 RUCM 用况模型，同时也适用于 RUCM4RT 用况模型(第 2 章)和 rtAspectRUCM 用况模型(第 3 章)，因此，在本章的行文中不再对它们进行严谨区分，而把它们通称为 RUCM 用况模型。

4.1 研究问题的提出

需求开发是系统开发生命周期中的第一个也是最关键的一个阶段，因为系统需求是后续开发活动(分析、设计、实现、测试)的驱动和依据，这对于复杂实时系统的开发尤为重要。随着用况建模成为工业实践中广泛使用的一种捕获、描述系统需求的方法[5,6]，因此如何确保用况模型达到相应的质量指标(如完整性、准确性、不模糊[72])是工业实践中的一个真实挑战。

需求评审，即软件评审[11]技术针对需求文档的一种特殊应用，是已被实证研究证实了并被广泛接受的一种有效技术，它可以较早的对需求中的缺陷进行识别和检测[12]。自从文献[15]对软件评审进行首次阐述，研究学者提出了各种不同的需求评审技术。基于检查列表的阅读技术[12](Checklist-based reading (CBR)[12])，通过设计一系列问题来指导需求评审员执行评审过程。基于缺陷的阅读技术[16] (Defect-based reading (DBR)[16])，针对某类特定的需求缺陷设计相应的检查列表以辅助评审人员执行评审过程。基于视角的阅读技术(Perspective-based reading methods (PBR)[17])，从不同的利益相关者的角度(如开发者、测试者的角度)进行相应的评审活动。基于使用的阅读技术[18] (Usage-based reading[18])，对用况进行优先化并从系统终端用户的视角进行需求评审。

需求评审是一个高度依赖于评审人员的过程，个人的经验、熟练程度都直接影响具体评审方法的有效性。针对不同的评审方法的有效性的对比分析，研究者经常给出不一致的结论[19]，从而无法为工业实践提供一种可靠而有效的评审技术。另外，在针对不同评审技术的有效性分析的实证研究中，研究学者都采用故障注入的方式预先创建相应的需求缺陷[19]。然而目前的实证研究实验中即缺少一种系统化而精确的需求缺陷分类，也没有给出一种系统化进行故障注入的方法，从而导致相应的实验几乎不能重现。

为了解决上述在比较不同需求评审技术中存在的问题，能够对不同的评审技术进行客观公正、有效可靠的评估分析，从而可以为工业实践推荐一种有效的需求评审技术。本章借鉴变异分析[44]，提出一种针对不同需求评审技术的成本-效益评估方法 MuRUCM。MuRUCM 通过严格遵循工业标准 IEEE Std. 830-1998[72]和最新的文献研究定义用况缺陷分类，通过严格遵循工业标准 HAZOP[73](Hazard and Operability Study)定义了一系列用况变异算子用以创建缺陷实例，最后 MuRUCM 给出了一组辅助规则用以创建缺陷生成策略从而可以系统化的进行故障注入。

4.2 MuRUCM 用况变异分析方法

图 21 提供了 MuRUCM 的概览图，它采用业务流程建模标记法(Business Process Model and Notion[111] (BPMN)) 标记方法进行说明，其中的一些概念和相应的关系在图 22 的概念模型中进行具体阐述。MuRUCM 的设计目标是提供一种系统化的方法为 RUCM 用况模型创建各种不同的缺陷，从而为评价各种基于 RUCM 的需求评审技术提供统一而一致的需求评审源，保证评价的客观性、公正性、可信性。如同变异测试(mutation testing)一样，MuRUCM 也是采用变异分数(mutation score)作为评价不同评审技术的效能指标。给定一个 RUCM 模型，一个 RUCM 变异算子的一次使用就会创建一个相应的 RUCM 变种，而这个 RUCM 变种包含一个特定的需求缺陷实例。此外，MuRUCM 还给出了一些规则用以辅助进行故障注入(即生成 RUCM 变种)。

如图 21 所示，整个 MuRUCM 方法可以分为两个过程。第一个过程是创建相应的 RUCM 模型，该过程通常由需求人员完成。该过程的制品是用况模型：用况规约和用况图，它为变异算子提供操作源。对于不同的 RUCM 扩展，相应的用况规约中可能会增

加一些特定领域的需求信息。例如RUCM4RT(第2章)会增加实时领域的特性需求信息。

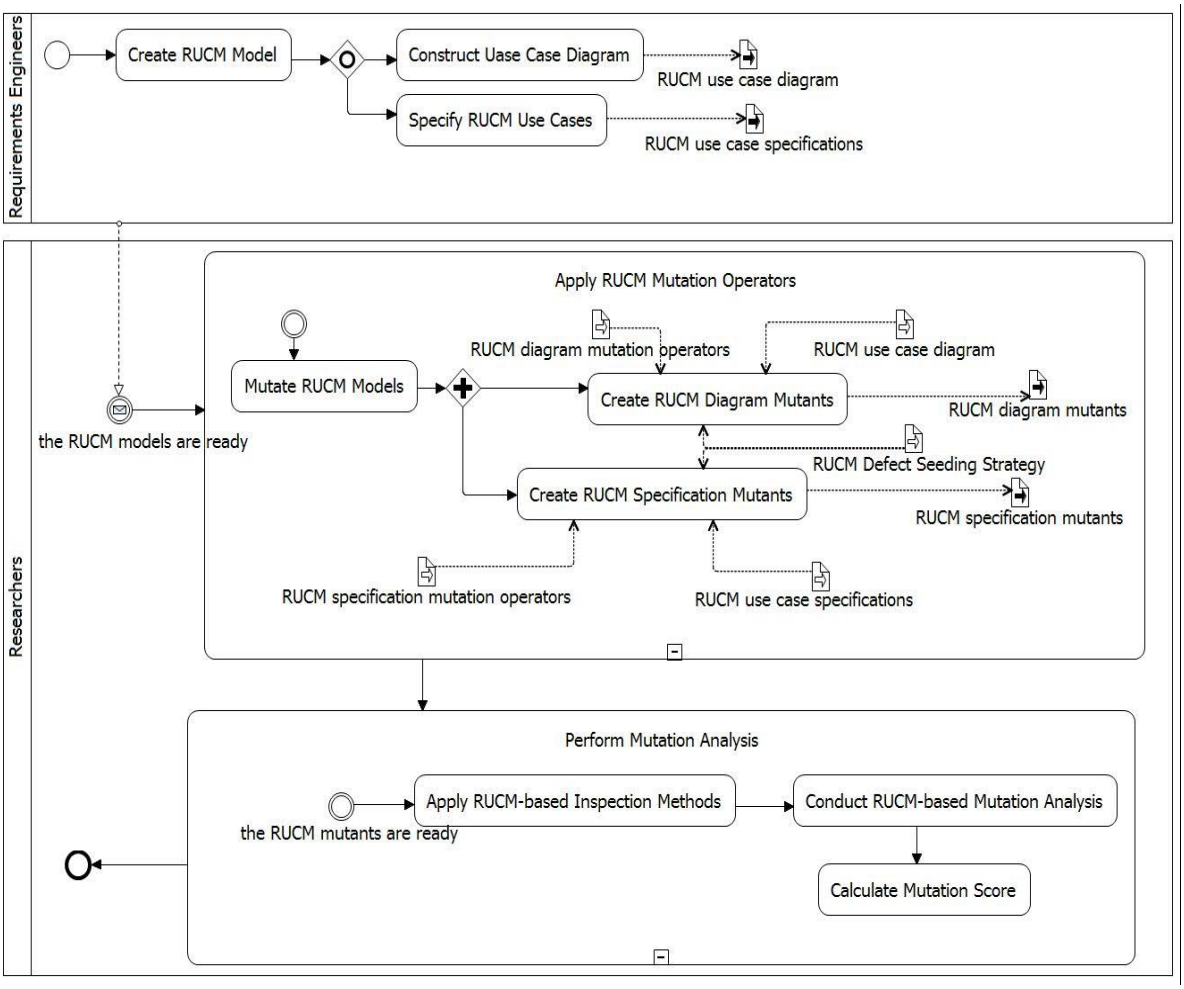

图 21 MuRUCM 方法的概览图(使用 BPMN 标记方法)

第二个过程是由研究人员进行，即变异分析过程，该过程又分为两个子过程：应用RUCM 变异算子和执行变异分析。应用 RUCM 变异算子的子过程：以第一个过程提供的 RUCM 模型作为操作源，依据 MuRUCM 提供的故障注入辅助规则，将选取的 RUCM 变异算子逐个的应用到 RUCM 模型中，从而产生一系列相应的 RUCM 变种。而其中的每一个 RUCM 变种就是某一类 RUCM 需求缺陷的一个具体实例。该子过程主要是产生不同类型的 RUCM 需求缺陷实例。

如图 22 所示，在 MuRUCM 方法中，需求缺陷是按照 RUCM 缺陷分类进行划分的，RUCM 缺陷分类是通过系统化的遵循工业标准 IEEE Std. 830-1998[72]创建的；每一个 RUCM 变异算子是通过系统化遵循工业标准 HAZOP[73,112]而创建的，HAZOP 是一个专门针对系统制品及其变异进行系统化分析的方法。具体讲，首先采用扩展的巴克斯-诺尔范式(Extended Backus-Naur Form[55](EBNF))对 UCMeta 进行定义，然后针对每一个

RUCM 元素(即 UCMeta 的一个具体模型元素)，系统化的使用每一个 HAZOP 的基本导航关键字进行分析，其中这些导航关键字都在标准 IEC 61882: 2001[73]有明确的定义。最后，依据 RUCM 缺陷分类定义相应的 RUCM 变异算子，即每一个 RUCM 变异算子，当且仅当使用它而产生的 RUCM 变种会包含一个 RUCM 缺陷分类中定义的需求缺陷的实例时，该 RUCM 变异算子才会被创建。简而言之，一个 RUCM 变异算子的一次使用必定会产生一个特定的 RUCM 需求缺陷的一个实例。

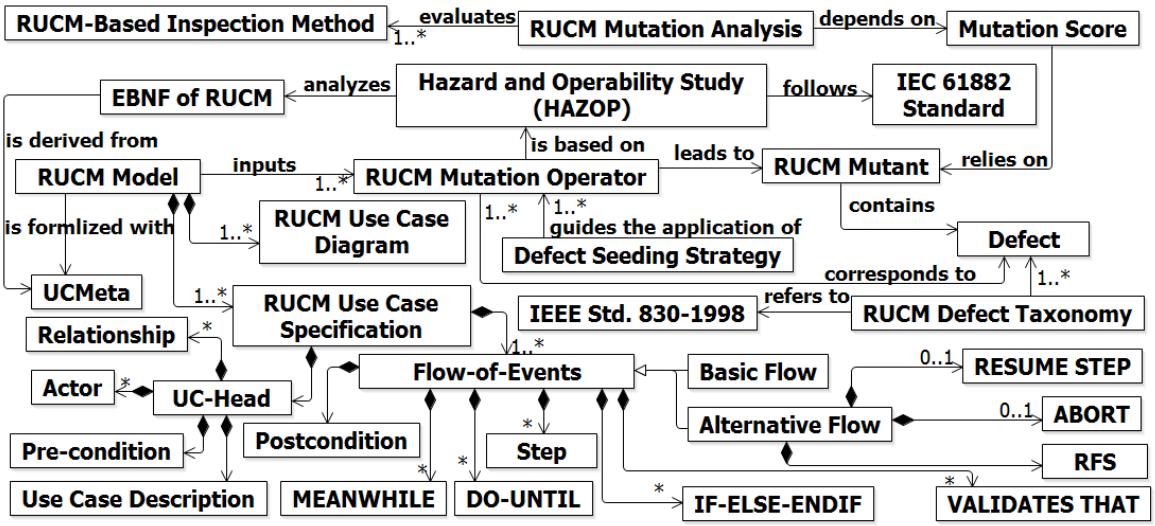

图 22 RUCM 变异分析的概念模型

通过使用 RUCM 变异算子和相应的 RUCM 缺陷注入辅助规则，创建 RUCM 变种以便对不同的 RUCM 需求评审技术进行评估。例如，在文献[113]中，通过使用 RUCM 变异算子创建了 541 个不同的 RUCM 需求缺陷实例。 每一个 RUCM 变种包含一个特定的 RUCM 缺陷的一个实例， 而 RUCM 用况图变种和 RUCM 用况变种分别代表针对 RUCM 用况图和 RUCM 用况而创建的需求缺陷。

执行变异分析的子过程：针对每一个特定 RUCM 需求评审技术，进行变异分析并计算相应的变异分数(Mutation Score)。基于上一个子过程创建的 RUCM 变种，研究人员可以计算相应的变异分数从而对当前正在分析的评审技术进行评估。例如，在 4.3 节中，为了对不同的 RUCM 覆盖策略进行评估，实验中通过计算相应的变异分数(图 29)进行分析。

本小节的文章结构如下：首先使用 EBNF 方法给出 UCMeta 的句法定义(4.2.1 节)，

其次详细阐述了 RUCM 缺陷分类的定义 (4.2.2 节)，然后论述了 RUCM 变异算子的生成(4.2.3 节)，最后给出了一组辅助规则用以创建相应的缺陷生成策略(4.2.4 节)。

4.2.1 RUCM 模型的句法构成

传统的变异分析依赖于对源程序的句法上的修改[114]。类似的，如图 23-图 24 所示，本节首先使用 EBNF[55] 范式对 RUCM 用况模型(即 UCMeta 模型元素)进行定义。

```
Use Case Model = Use Case Diagram, Use Case Specification;
Use Case Diagram = UCD-UseCase, UCD-Actor, UCD-Relationship;
UCD-UseCase = {UCD::Use Case}-;
UCD-Actor = {UCD::Actor}-;
UCD-Relationship = {Association}-, {UCD-Generalization | UCD-Include |
UCD-Extend}
Use Case Specification = UC-Head, Basic Flow, {Alternative Flow};
UC-Head = Use Case, Pre-condition, Actor, {Dependency},
{Generalization};
Use Case = UC-Name, UC-BriefDescription;
UC-Name = use case name;
UC-BriefDescription = brief description of the use case;
Actor = {Prim-Actor}-, {Sec-Actor};
Prim-Actor = primary actor name;
Sec-Actor = secondary actor name;
Dependency = Include | Extend;
Generalization = {UC-Name};
Pre-condition = precondition description of the use case;
Include = "INCLUDE USE CASE", {UC-Name}-;
Extend = "EXTENDED BY USE CASE", {UC-Name}-;
Basic Flow = {Flow-name}, {Action Step}-, {Condition Step | MEANWHILE
 | VALIDATES THAT}, Post-condition;
Alternative Flow = Flow-name, RFS, {Action Step}-, {Condition Step |
 MEANWHILE}, (RESUME STEP | ABORT), Post-condition;
Flow-name = the name of a RUCM flow;
Action Step = RUCM action sentence;
MEANWHILE = Action Step, "MEANWHILE", Action Step;
VALIDATES THAT = "VALIDATES THAT", Action Step;
Condition Step = IF-ELSE-ENDIF | DO-UNTIL;
RFS = "RFS", flow-name, Step-index;
Step-index = int, {"-", int};
```

图 23 RUCM 模型元素的 EBNF 定义(一)

```
IF-ELSE-ENDIF = IF-THEN-ENDIF | IF-THEN-ELSE-ENDIF | IF-THEN-ELSEIF-
 THEN-ENDIF | ELSE-ENDIF | ELSEIF-THEN-ENDIF;
IF-THEN-ENDIF = "IF", condition, "THEN", {Action step}-, "ENDIF";
IF-THEN-ELSE-ENDIF = "IF", condition, "THEN", {Action step}-, "ELSE",
 {Action step}-, "ENDIF";
IF-THEN-ELSEIF-THEN-ENDIF = "IF", condition, "THEN", {Action step}-,
 "ELSEIF", condition, "THEN", {Action step}-, "ENDIF";
ELSE-ENDIF = "ELSE", {Action step}-, "ENDIF";
ELSEIF-THEN-ENDIF = "ELSEIF", Condition, "THEN", {Action step}-,
 "ENDIF";
DO-UNTIL = "DO", {Action step}-, "UNTIL", condition;
RESUME STEP = "RESUME STEP", Step-index;
Condition = RUCM condition sentence;
ABORT = "ABORT";
Post-condition = post-condition description of the use case;
RUCM action sentence = a RUCM Sentence describing system action;
RUCM condition sentence = a RUCM Sentence describing conditions;
RUCM Sentence = a descriptive sentence following RUCM writing rules
(e.g., simple present tense, forbidden usage of adverbs, adjectives,
pronouns, synonyms and negatives);
int = digital-'0', {digital};
digital = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';
```

图 24 RUCM 模型元素的 EBNF 定义(二)

本文采用一种更加灵活的方式使用 EBNF 描述 RUCM 用况模型。例如，EBNF 表达式中的终结符可以是一个自然语言表达式，如'UC-Name'，也可以是一个 UCMeta 模型元素，如 RUCM *Sentence*。图 23-图 24 中的 EBNF 表达式是对 UCMeta 的形式化定义，因为 UCMeta 是 RUCM 采用的形式化策略，它是 RUCM 模型的具体载体和管理者，每一个 RUCM 建模元素都会被实例化为一个对应的 UCMeta 元素，UCMeta 的角色如同程序设计语言(如 Java、C)的语法规范。RUCM 变异算子的操作效果最终都要通过 UCMeta 实现，在本章的后续行文中，使用 EBNF 表达式表示一个 RUCM 建模元素的定义。

4.2.2 RUCM 缺陷分类

本节基于工业标准 IEEE Std. 830-1998[72]和最新的文献研究定义 RUCM 的缺陷分类。

4.2.2.1 定义

本文遵循工业标准 IEEE Std. 830-1998[72]定义 RUCM 缺陷分类。标准 IEEE Std. 830-

1998[72] 针对软件需求规约(SRS)中可能出现的问题，给出了相应的缺陷分类，本文对这些分类定义进行系统化研究，并将它们应用 RUCM 用况模型中。此外，通过文献调研，本文还从[115,116]和[6,117,116]中分别引入另外两种需求缺陷类型。

不正确性 Incorrectness (C1)。在 IEEE Std. 830-1998[72]中，一个 SRS 是正确的，“当且仅当每一个陈述的需求都是软件系统必须要实现的”。那些没有给出系统的真实需要的 SRS 应该被认为是不正确的。这类错误直接反映了对用户意图的误解，在 RUCM 方法中，需要用户对 RUCM 用况规约进行确认或者将 RUCM 用况规约和其他的文档(如项目合同等)进行对比而发现。

不完整性 Incompleteness (C2)。如 IEEE Std. 830-1998[72]所陈述的，一个 SRS 是完整的当且仅当它包含了以下的所有元素：1) 所有重要的需求(如功能需求、性能需求、设计约束、额外属性)，2) 定义出系统在各种环境下针对各种输入的反馈，3) 给出 SRS 中使用的各个表、图的完整标签和索引。在 RUCM 用况模型的环境下，任何一个 RUCM 模型元素(如 *Action Step*)的缺失都会导致 RUCM 模型的不完整性。例如，缺少一个用况则直接导致对系统规约说明的不完整。RUCM 自身通过设计不同类型的事件流，希望在一定程度上能够辅助需求人员进行需求描述时减少这种不完整性。

不一致性 Inconsistency (C3)。在 IEEE Std. 830-1998[72]中，一致性被解释为：陈述的各个需求项之间是一致的，彼此之间不应该存在冲突。对 RUCM 而言，本文关注用况图和用况规约之间的一致性，以及一个用况规约内部不同的需求描述语句之间的一致性，例如，一个事件流中的不同的需求描述语句的描述内容之间可能会存在冲突。

模糊性 Ambiguity (C4)。一个 SRS 是不模糊的当且仅当每一个陈述的需求项都有唯一的一种解释[72]。为了避免这种错误，它要求需求人员系统化的使用一致性的术语对 RUCM 用况模型进行规约说明。此外，RUCM 定义了 26 条使用自然语言的限制规则用以最大程度的减少用况模型中的不一致性。

难以理解 Incomprehensibility (C5)。这中缺陷分类是从文献[6,117,116]中借鉴来的，它主要强调一个用况模型应该被很容易的理解。在 RUCM 中，相应的 RUCM 用况图及用况规约中的每一个需求描述语句都应该可以被需求人员、设计人员、开发人员、测试人员很容易的理解。

不可测性 Intestability (C6)。一个需求可以被验证，当且仅当“存在一个有限的过程，

一个人或机器可以判断系统是否可以满足需求[72]”。RUCM 的固有机制(即前置条件 *Precondition* 和后置条件 *Post-conditions*)用于帮助对每一个事件流进行验证。此外，RUCM 模型已被用于创建测试用例，如[118,119]。当定义 RUCM 缺陷分类时，本文从测试者的角度出发，将焦点放在 *Preconditions* 和 *Post-conditions* 的需求描述中。

不可修改性 Unmodifiability (C7)。一个 SRS 是可修改的，当且仅当对它的任何修改都可以 “方便的进行，同时保持它的结构和形式是完整的、一致的[72]”。在 RUCM 环境下，一个共用的行为应该被独立建模并可复用(如通过«*Include*»或«*Extend*»实现)，而不是被散落在不同的用况中。

不可实现 Infeasibility (C8)。这类需求缺陷是从系统开发者的视角定义那些无法实现的需求。

不必要需求 Over-Specification (C9)。这种分类是从[115,116]借鉴来的。它说明用况模型中存在一些不相关的需求信息。针对 RUCM，它要求 RUCM 模型特别是 RUCM 用况规约中不应该描述额外不必要的信息。

4.2.2.2 RUCM 缺陷类别

4.2.2.1 节给出的缺陷定义，不仅可以作为评估 RUCM 模型的质量指标，而且可以作为一种缺陷分类方式。本节将详细论述 RUCM 缺陷分类，基于 4.2.2.1 节的缺陷定义，本节系统化的定义了 104 种具体的 RUCM 需求缺陷(表 12-表 13)。

```
Defect = Defect Prefix, Defect ID
Defect Prefix = Defect Category, Defect Element;
Defect Category = C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9;
Defect Element = AR | UC | R | H | F | UCM | UCS;
Defect ID = '1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';
AR = Actor;
UC = Use Case;
R = relationship between two model elements;
H = the head of a use case specification;
F = flow-of-events of a use case specification;
UCM = use case model;
UCS = use case specification;
```

图 25 RUCM 缺陷类型的命名机制(使用 EBNF 定义)

针对每一个 RUCM 模型元素(使用 EBNF 范式进行定义)，分别从 9 个不同的视角(4.2.2.1 节中 9 种不同的分类方式)对其进行分析，一旦某个缺陷分类可以与当前 RUCM 模型元素匹配，那么就定义一种相应的 RUCM 缺陷。如表 12-表 13 所示，这些缺陷既包含用况模型中的结构型缺陷(如用况之间的错误关系)也有用况规约中的语义缺陷(如 *Ambiguity* (4.2.2.1 节))。对于每一个 RUCM 缺陷，本文采用图 25 的命名机制。

如表 12-表 13 所示，本文中定义的缺陷分类可以划分为两大类：针对用况图的缺陷，针对用况规约的缺陷。其中大部分的缺陷定义都是针对用况规约的，因为用况规约描述了系统行为的细节信息。不一致性 *Inconsistency* (C3)不适用于用况图，本文目前并不考虑不同抽象层次创建的用况图之间的一致性。然而，用况图与相应的用况规约之间的一致性(C3UCM1)，以及用况规约中各个元素的一致性是本文的考虑重点。不可测性 *Intestability* (C6)和不可实现 *Infeasibility* (C8)也同样不适用于用况图，因为它们并没有包含系统行为的具体细节。在本文中，用况规约又进一步细分为两个组成部分：表头 UC-Head 和 RUCM 事件流(Flow-of-Events)。UC-Head 元素(如图 4)包含用况名字、首要参与者、辅助参与者、用况依赖关系 («*Include*»、«*Exclude*»)、用况泛化关系 *Generalization* 等，这些元素构成了 RUCM 用况模板的第一个组成部分。 UC-Head 的缺陷都是针对这些元素设计的，例如，缺失一个首要参与者(C2H1)。由于 UC-Head 并没有描述系统的动作行为，因此 *Infeasibility* (C8)并适用于这些元素。用况前置条件 Precondition 是 UC-Head 的一个元素，因此 *Intestability* 仍然适用(即 C6H1)。

用况规约中的事件流详细描述了系统与参与者之间的交互以及相应的动作行为，针对事件流的缺陷定义需要考虑全部的 9 种缺陷分类(C1-C9)。相关的 RUCM 元素包含：基本流(Basic Flow)、备选流(Alternative Flow)、RUCM 动作语句(Action Step)、RUCM 条件语句(Condition Step)、后置条件(Post-condition)、分支参考语句 RFS (创建一个备选流的分支语句)、含有 RUCM 关键字 (如 ABORT)的语句、事件流中的语句序列。有些 RUCM 缺陷 (即 C2UCS1、C3UCS1-C3UCS4) 针对整个用况规约进行定义，有的缺陷 (即 C7UCS1) 涉及事件流中的多个模型元素。缺陷 C3UCM1 考虑用况图和用况规约之间的一致性，缺陷 C7UCM1 考虑不同用况之间的关系，因此这两类缺陷是针对整个用况模型进行定义的。

表 12 RUCM 缺陷分类(总共 104 种不同类型)

Category	Use Case Diagram (UCD)			Use Case Specification (UCS)	
	Actor	Use Case	Relationship	UC-Head	Flow-of-Events
C1	**C1AR1:** Incorrect Actor	**C1UC1:** Incorrect Use Case	**C1R1:** Incorrect Generalization between Actors **C1R2:** Incorrect Generalization between Use Cases **C1R3:** Incorrect Include between Use Cases **C1R4:** Incorrect Extend between Use Cases **C1R5:** Incorrect Association between Actor and Use Case	**C1H1:** Incorrect Primary Actor **C1H2:** Incorrect Secondary Actor **C1H3:** Incorrect Brief Description **C1H4:** Incorrect Pre-condition **C1H5:** Incorrect Include **C1H6:** Incorrect Extend **C1H7:** Incorrect Generalization **C1H8:** Incorrect Resource **C1H9:** Incorrect Period **C1H10:** Incorrect TimeCost **C1H11:** Incorrect ObservationVariable **C1H12:** Incorrect Constraint	**C1F1:** Incorrect Basic Flow **C1F2:** Incorrect Alternative Flow **C1F3:** Incorrect numbering of steps **C1F4:** Incorrect Post-condition **C1F5:** Incorrect branching of alternative flow **C1F6:** Incorrect merging of alternative flow **C1F7:** Incorrect logical relationship
C2	**C2AR1:** Missing Actor	**C2UC1:** Missing Use Case	**C2R1:** Missing Generalization between Actors **C2R2:** Missing Generalization between Use Cases **C2R3:** Missing Include **C2R4:** Missing Extend **C2R5:** Missing Association between Actor and Use Case	**C2H1:** Missing Primary Actor **C2H2:** Missing Secondary Actor **C2H3:** Missing Brief Description **C2H4:** Missing Pre-condition **C2H5:** Missing Include **C2H6:** Missing Extend **C2H7:** Missing Generalization **C2H8:** Missing Resource **C2H9:** Missing Period **C2H10:** Missing TimeCost **C2H11:** Missing ObservationVariable **C2H12:** Missing Constraint	**C2F1:** Missing Alternative Flow **C2F2:** Missing Step **C2F3:** Missing RFS **C2F4:** Missing post-condition **C2F5:** Missing RESUME **C2F6:** Missing ABORT **C2F7:** Missing logical relationship
				C2UCS1: Missing UCS	
C3	–			**C3H1:** Actor is inconsistent with its behavior in use cases. **C3H2:** Brief Description is inconsistent with the design intent. **C3H3:** Pre-condition is inconsistent with the design intent.	**C3F1:** Basic Flow is inconsistent with its expected goal. **C3F2:** Alternative Flow is inconsistent with its expected goal. **C3F3:** Inconsistent definition of references flows in an alternative flow. **C3F4:** Post-condition is inconsistent with its expected goal. **C3F5:** The numbing of steps is inconsistent.
				C3UCS1: UCS is inconsistent with the design intent. **C3UCS2:** Inconsistent terminologies used in the UCS. **C3UCS3:** Description in the UCS conflicts with another one. **C3UCS4:** UCS should be documented in a consistent level of abstraction.	
	C3UCM1: UCD is inconsistent with its corresponding UCSs.				
C4	**C4AR1:** The actor name does not reflect its role.	**C4UC1:** The use case name does not reflect its goal.	–	**C4H1:** Ambiguous Brief Description **C4H2:** Ambiguous Pre-condition	**C4F1:** Ambiguous sentence in Basic Flow. **C4F2:** Ambiguous sentence in Alternative Flow. **C4F3:** Ambiguous Post-condition.
				C4UCS1: Not using the simple present tense throughout the UCS causes ambiguity. **C4UCS2:** Not avoiding the usage of adverbs, adjectives, pronouns, synonyms and negatives causes ambiguity.	
C5	**C5AR1:** Incomprehensible actor name	**C5UC1:** Incomprehensible use case name	–	**C5H1:** Incomprehensible Brief Description **C5H2:** Incomprehensible Pre-condition	**C5F1:** Incomprehensible sentence of Flow-of-Events. **C5F2:** Incomprehensible Post-condition.
C6	–			**C6H1:** The Pre-condition can never be satisfied.	**C6F1:** The behavior of Flow-of-Events can never be measured. **C6F2:** The Post-condition can never be measured. **C6F3:** The Flow-of-Events should be terminated reasonably.
C7	–			**C7UCS1:** Alternative flows should be separated from Basic Flow.	
	C7UCM1: A shared system functionality should be specified as a separate use case and associated to others.				
C8	–			–	**C8F1:** The behavior described in Flow-of-Events cannot be implemented.

表 13 RUCM 缺陷分类(总共 104 种不同类型)-续

Category	Use Case Diagram (UCD)			Use Case Specification (UCS)	
	Actor	Use Case	Relationship	UC-Head	Flow-of-Events
C9	**C9AR1:** Superfluous actor	**C9UC1:** Superfluous use case	**C9R1:** Superfluous Generalization between Actors **C9R2:** Superfluous Generalization between Use Cases **C9R3:** Superfluous Include **C9R4:** Superfluous Extend **C9R5:** Superfluous Association between Actor and Use Case	**C9H1:** Superfluous secondary actor **C9H2:** Superfluous sentences in Brief Description **C9H3:** Superfluous sentences in Pre-condition **C9H4:** Superfluous Include **C9H5:** Superfluous Extend **C9H6:** Superfluous Generalization (with other use cases)	**C9F1:** Superfluous alternative flow **C9F2:** Superfluous step **C9F3:** Superfluous sentence in Post-condition.

4.2.3 RUCM 变异算子

传统的变异分析依赖于程序变种(即对源程序的句法修改[114])，而变异算子扮演的角色就是如何从源程序产生程序变种的转换规则。本文采用另外的策略定义 RUCM 变异算子，因为 RUCM 模型实际上是软件需求，因此与传统的变异算子生成方式有显著的不同。本节的内容组织如下：4.2.3.1 节阐释了选取的 HAZOP 导航关键字，4.2.3.2 节详细论述了生成 RUCM 变异算子的机制，4.2.3.3 节报告了所有生成的 RUCM 变异算子。

4.2.3.1 HAZOP 的剪裁使用

HAZOP[73]是在航空工业实践中广泛采用的一种分析技术，通过对一个过程、操作、设计、个人错误进行系统化分析从而识别出相应的危害，例如航空工业中开发安全关键系统时使用 HAZOP 进行风险分析[73]。“系统设计目标”是 HAZOP 分析的基准线而且它应该尽可能的准确和完备[73]，在本文的环境下，“系统设计目标”就是用户的真实需求。一个成功的 HAZOP 应用依赖于两方面[73]：1) 识别出系统的构成元素及其相应的属性、参数，2) 一组导航关键字，其中的每一个导航关键字都要独立的应用到每一个系统元素的每一个参数，从而识别出那些未曾期望的但是合理的系统设计的偏差。在工业标准 IEC 61882 standard[73]中定义了基本的 HAZOP 导航关键字：NO、MORE、LESS、AS WELL AS、PART OF、REVERSE、OTHER THAN。本文采用这些关键字对 RUCM 模型元素进行分析，表 14 给出了它们在 RUCM 语境下的具体定义。

表 14 MuRUCM 中选用的 HAZOP 导航关键字

Guide word	Definition
NO	An intended RUCM model element is not captured. A RUCM modeling intent is not achieved.
MORE	Quantitative increase in a quantifiable RUCM element.
LESS	Quantitative decrease in a quantifiable RUCM element.
AS WELL AS	Spurious RUCM model elements or behavior are included in a RUCM model.
PART OF	Incomplete RUCM modeling intent is achieved. Incomplete model element or behavior is captured.
REVERSE	A logical opposite of the RUCM modeling intent, behavior or model element is specified.
OTHER THAN	A RUCM model is substituted by unintended/incorrect behavior, intent or model element.

4.2.3.2 变异算子的产生机制

本文采用 7 个基本 HAZOP 导航关键字对 RUCM 元素进行分析，并识别出那些可以使用 HAZOP 导航关键字的 RUCM 建模元素。本文的策略中，将每一个 HAZOP 导航关键字与每一个 RUCM 元素进行匹配映射，通过分析当前匹配操作的负影响来创建相应的 RUCM 变异算子。图 26 描绘了这种机制，从中可以发现该生成机制是 5 维的：

1) 7 个基本的 HAZOP 导航关键字：NO、MORE、LESS、AS WELL AS、PART OF、REVERSE、OTHER THAN ，即 *x* 轴。

2) 9 种不同 RUCM 缺陷分类(如 *Incorrectness (C1)*)，即 *y* 轴。

3) RUCM 模型元素，即 *z* 轴。

4) 6 种模型编辑操作：ADD(添加操作)、DEL(删除操作)、SWAP(对 RUCM 模型中的两个元素实施兼容性的交换)、REP(使用一个不在当前 RUMC 模型中元素对当前 RUCM 模型中存在的一个元素实施兼容性的替换)、ICR(定量的增加)、DEC(定量的减少))。这些操作是 RUCM 变异算子命名的第一个构成部分 (如‘DEL-AF-C2F1’中的‘DEL’)。

5) RUCM 需求缺陷。当把一个编辑操作作用到一个 RUCM 模型元素时，相应的后果必须使用 RUCM 需求缺陷进行确定。RUCM 需求缺陷是 RUCM 变异算子命名的第三个构成部分 (如‘DEL-AF-C2F1’中的‘C2F1’)。

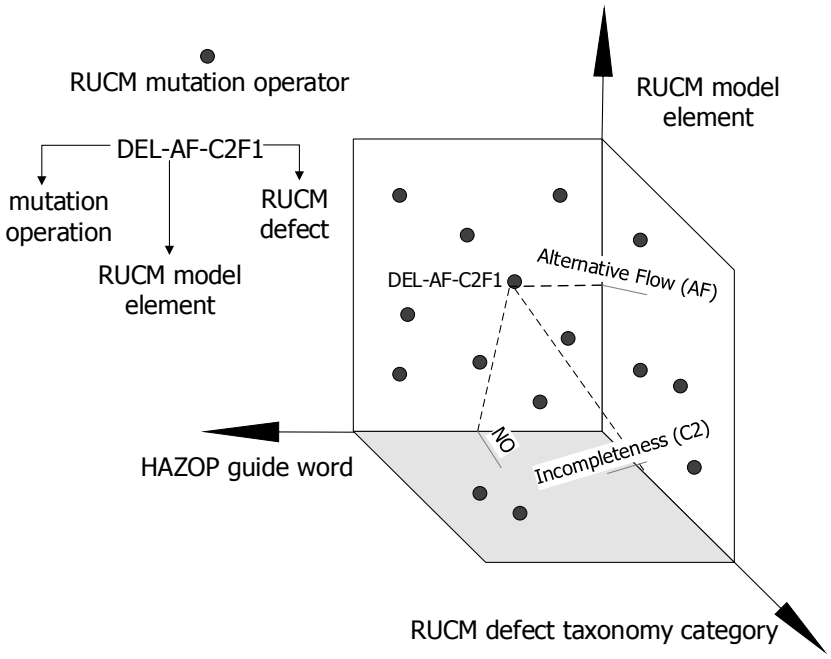

图 26 RUCM 变异算子的生成机制

通过采用以上机制，可以确保每一个 RUCM 变异算子的一次使用可以产生一个特定的 RUCM 缺陷的一个实例。其中每个 RUCM 变异算子由三个部分构成：一个编辑操作，一个可以实施该编辑操作的 RUCM 模型元素，一种由该变异算子导致的 RUCM 缺陷类别。给定一个 RUCM 模型元素，HAZOP 导航关键字的使用可以系统化的将一个编辑操作与一个 RUCM 缺陷分类中定义的缺陷进行关联。 例如，图 26 中变异算子 ‘DEL-AF-C2F1’的创建过程如下：首先，识别出变异算子可以操作的 RUCM 模型元素，此时选定的是 RUCM 备选流，即图 26 中的 z 轴。然后，使用每一个 HAZOP 关键字对前面选取的 RUCM 模型元素(即 RUCM 备选流)进行分析。当把 HAZOP 的 NO 关键字应用到当前选定的 RUCM 备选流时，相应的解释是：选取一个编辑操作，它可以导致原 RUCM 模型丢失一个备选流，即发生一个 NO 偏差(图 26 中的 x 轴)，此时会选定 DEL 编辑操作。最后，将编辑操作 DEL 实施到选定的一个 RUCM 备选流，即从 RUCM 模型中将选定的那个备选流删除掉，这直接导致了 RUCM 模型缺失了一个备选流(C2F1)，相应的也就创建了 C2 (*Incompleteness*)类型的缺陷(如图 26 中的 y 轴所示)。最终，RUCM 变异算子‘DEL-AF-C2F1’就被创建。

在本文中，针对每一个 HAZOP 关键字，系统化的分析它可以适用于哪个 RUCM 模型元素。例如，图 23 所示，元素‘Use Case’的 EBNF 表达式是‘Use Case = UC-Name, UC-

BriefDescription;’，其中 UC-Name 表示用况的名字，因此 HAZOP 关键字 OTHER THAN 可以应用到这个元素，即通过修改 UC-Name 的字面值使得当前用况成为 RUCM 模型中的另一个用况。RESUME STEP 和 ABORT 是 RUCM 模型中含有这两个 RUCM 关键字的动作语句。由于这两个 RUCM 元素只可以互斥的应用到 RUCM 备选流中，因此可以对它们使用 HAZOP 关键字 OTHER THAN，即对它们进行相互替换。针对 RUCM 的事件流(Flow-of-Events)，HAZOP 关键字 PART OF 和 AS WELL AS 都可以使用，因为‘Flow-of-Events’是一个 RUCM Step 容器(图 23-图 24)。

每一个 HAZOP 关键字对一个 RUCM 元素的应用都是对 RUCM 模型的一次操作。本文定义了以下 6 种模型编辑操作：添加一个模型元素(ADD)，删除一个模型元素(DELETE)，交换两个模型元素(SWAP)，替换一个模型元素(REP)，增加一个模型元素的数据值(ICR)，减少一个模型元素的数据值(DEC)。

4.2.3.3 RUCM 变异算子

***变异算子的命名机制*:**

表 15-表 19 报告了所有的 RUCM 变异算子。本文针对变异算子设计图 27-图 28 中的命名机制。

```
    RUCM mutation operator = Operation, -, Element, -, {Defect}-;
    Operation = ADD | DEL | SWAP | REP | ICR | DEC;
    Element = UC | AR | INC | EXD | ASSO | UCS | UCN | ARN | PAR | SAR
| BD | GA | GUC | SenBD | PreC | SenPreC | AF | SenAF | BF | SenBF |
PostC | SenPostC | AS | RFS | ABORT | IFELSE | DO | VLD | MW |RFSsi |
RFSflow | toABT | RES | toRES | IFELSE | IFELSEcs | IFELSEas | DOcs |
DOas | RCE | PER | PERvalue | TC | TCvalue | OBV | OBVrfs | TimC |
TimCvalue | ResC | ResCvalue;
    Defect = Defect Prefix, Defect ID
    Defect Prefix = Defect Category, Defect Element;
    Defect Category = C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9;
    Defect Element = AR | UC | R | H | F | UCM | UCS;
    Defect ID = ‘1’|‘2’|‘3’|‘4’|‘5’|‘6’|‘7’|‘8’|‘9’;
    UC = Use Case;
    AR = Actor;
    INC = Include;
    EXD = Extend;
```

图 27 RUCM 变异算子的命名机制

ASSO = Association; **PAR** = primary actor;
SAR = secondary actor; **UCS** = use case specification;
UCN = use case name; **UCM** = use case model;
BD = use case brief description; **ARN** = Actor name;
GAR = generalization between actors;
GUC = generalization between use cases;
SenBD = sentence in BD; **PreC** = pre-condition;
SenPreC = sentence in PreC; **BF** = Basic Flow;
SenBF = sentence in BF; **AF** = Alternative Flow;
SenAF = sentence in AF; **PostC** = post-condition;
SenPostC = sentence in PostC; **AS** = Action step;
VLD = VALIDATES THAT; **MW** = MEANWHILE; **RFS** = sentence RFS;
RFSsi = step index of RFS;
RFSflow = reference flow name of RFS;
ABORT = sentence ABORT;
toRES = changing ABORT to RESUME STEP;
RES = RESUME STEP;
toABT = changing RES to ABORT;
IFELSE = IF-SELSE-ENDIF;
IFELSEcs = condition sentence in IF-ELSE-ENDIF; **IFELSEas** = Action Step in IF-ELSE-ENDIF;
DO = DO-UNTIL;
DOcs = condition sentence in DO-UNITL;
DOas = Action step in DO-UNITL;
RCE = resource element of real-time use case;
PER = periviod element of periodic use case;
PERvalue = time value of PER;
TC = time cost of real-time use case;
TCvalue = time value of TC;
OBV = observation variables of real-time use case;
OBVrfs = RFS of OBV;
TimC = time constriants of real-time use case;
TimCvalue = time value of TimC;
ResC = resource constraint of real-time use case;
ResCvalue = resource value of ResC, if possible;

图 28 RUCM 变异算子的命名机制(续)

一个HAZOP导航关键字与一个RUCM模型元素的组合可能会引入不同类型的RUCM缺陷，从而会产生不同的RUCM变异算子。为了节省空间，表15-表19中，采用‘{ }’包含相应的多个RUCM变异算子。例如，REP-UCN-{C1U1, C4U1, C5U1}(表

15)表示了3个RUCM变异算子：REP-UCN-C1U1、REP-UCN- C4U1、REP-UCN-C5U1。

***用况图的变异算子*:**

表15展示了针对RUCM用况图的所有29个变异算子。例如SWAP-INC-C1R3，它表示对RUCM模型元素*Include*使用HAZOP关键字REVERSE，即将*Include*关系中的两个用况的角色进行互换(包含用况变为被包含的用况，被包含的用况变为包含用况)，从而导致*Incorrectness*缺陷类别中的C1R3。HAZOP导航关键字‘MORE’和‘LESS’不可以应用到用况图，因为这两个关键字只可以作用到可以进行量化操作的RUCM模型元素中。‘AS WELL AS’也不可以作用到用况图中，因为用况图中出现的RUCM元素都不再包含其他的元素。

表15 RUCM用况图的变异算子(29种)

	NO	AS WELL AS	REVERSE	OTHER THAN
UCD::Use Case	DEL-UC-C2UC1	ADD-UC-C9UC1	-	REP-UCN-{C1UC1, C4UC1, C5UC1}
UCD::Actor	DEL-AR-C2AR1	ADD-AR-C9AR1	-	REP-ARN-{C1AR1, C4AR1, C5AR1}
UCD::Include	DEL-INC-C2R3	ADD-INC-C9R3	SWAP-INC-C1R3	REP-INC-C1R3
UCD::Extend	DEL-EXD-C2R4	ADD-EXD-C9R4	SWAP-EXD-C1R4	REP-EXD-C1R4
UCD::Generalization	DEL-AR-C2R1 DEL-UC-C2R2	ADD-AR-C9R1 ADD-UC-C9R2	SWAP-GAR-C1R1 SWAP-GUC-C1R2	REP-GAR-C1R1 REP-GUC-C1R2
UCD::Association	DEL-ASSO-C2R5	ADD-ASSO-C9R5	-	REP-ASSO-C1R5

***用况规约的变异算子*:**

表16中的34个RUCM变异算子是针对RUCM用况规约中的表头UC-Head元素设计的。如同RUCM用况图一样，HAZOP导航关键字MORE和LESS对UC-Head也不适用。由于用况规约的简要描述*brief description*和前置条件*precondition*可以包含一个或多个RUCM *Step*，HAZOP导航关键字PART OF可以作用到UC-Head。

表 16 RUCM 用况规约表头(UC-Hea)的变异算子(34 种)

	NO	AS WELL AS	PART OF	REVERSE	OTHER THAN
UCS::Use Case	-	-	-	-	-
UCS::Actor	DEL-PAR-C2H1 DEL-SAR-C2H2	ADD-SAR-C9H1	-	SWAP-AR-{C1H1, C1H2}	REP-PAR-C1H1 REP-SAR-C1H2
UCS::Brief Description	DEL-BD-C2H3	ADD-SenBD-C9H2	DEL-SenBD-C1H3 REP-SenBD-C1H3	-	REP-BD-{C1H3, C3H2, C4H1, C5H1}
UCS::Include	DEL-INC-C2H5	ADD-INC-C9H4	-	-	REP-INC-C1H5
UCS::Extend	DEL-EXD-C2H6	ADD-EXD-C9H5	-	-	REP-EXD-C1H6
UCS::Generalization	DEL-GUC-C2H7	ADD-GUC-C9H6	-	-	REP-GUC-C1H7
UCS::Precondition	DEL-PreC-C2H4	ADD-SenPreC-{C9H3, C6H1}	DEL-SenPreC-{C1H4, C6H1}	-	REP-PreC-{C1H4, C3H3, C4H3, C5H2, C6H1}

针对 RUCM 的事件流 Flow-Of-Events，通过 HAZOP 分析设计了 54 个 RUCM 变异针对 RUCM 事件流(FlowOfEvents)的变异算子，在表 18 中进行报告。如同 RUCM 用况图和 UC-Heads，HAZOP 导航关键字 MORE 和 LESS 也不适应于 RUCM 的事件流 Flow-Of-Events。表 17 列举了 28 个变异算子，它们可以直接产生实时特性相关的需求缺陷。例如，变异算子‘ICR-PERvalue-C1H9’的使用会导致当前实时用况的时间周期的数值增大，这样就会创建一个 C1H9 类型的缺陷实例。表 19 报告了 74 个 RUCM 变异算子，它们都是针对用况规约中那些含有 RUCM 关键字(如 VALIDATES THAT)的语句而设计的。由于 RUCM 关键字‘RFS’和‘RESUME STEP’都需要 RUCM 语句序号(一个表示事件流中第几个 RUCM 语句(Step)的数字)，因此 HAZOP 导航关键字 MORE 和 LESS 可以应用到这两个 RUCM 元素。相应的会采用编辑操作 ICR 和 DEC 对 RFS 和 RESUME STEP 中包含的 RUCM 语句序号进行增大和减小的操作。

表 18 RUCM 事件流(FlowOfEvents)的变异算子(54 种)

	NO	AS WELL AS	PART OF	REVERSE	OTHER THAN
Flow of Events	-	ADD-AF-C9F1	DEL-AF-C2F1	SWAP-AF-{C1F5, C3F2}	-
Basic Flow	-	ADD-SenBF-C9F2	DEL-SenBF-{C1F1, C2F2, C3F1, C3F5, C6F1, C6F3, C8F1}	SWAP-SenBF-{C1F1, C1F3, C3F1, C6F1, C6F3, C8F1}	REP-BF-C1F1
Alternative Flow	DEL-AF-C2F1	ADD-SenAF-C9F2	DEL-SenAF-{C1F2, C2F2, C3F2, C3F5, C6F1, C6F3, C8F1}	SWAP-SenAF-{C1F2, C1F3, C3F2, C6F1, C6F3, C8F1}	REP-AF-C3F2
Post-condition	DEL-PostC-C2F4	ADD-SenPostC-{C9F3, C6F2}	DEL- SenPostC-C1F4 REP- SenPostC-C6F2	-	REP-PostC-{C1F4, C3F4, C4F3, C5F2, C6F2
Action Step	DEL-AS-C2F2	-	-	-	REP-AS-{C4UCS1, C4UCS2, C4F1, C4F2, C5F1, C6F1, C6F3, C6F3}

表 17 实时特性相关的变异算子(28 种)

	NO	MORE	LESS	PART OF	OTHER THAN
UCS::Resource	DEL-RCE-C1H8	-	-	DEL-RCE-C2H8	REP-RCE-C1H8
UCS::Period	DEL-PER-C1H9	ICR-PERvalue-C1H9	DEC-RERvalue-C1H9	DEL-PER- C2H9	REP-RES-C1H9
UCS::TimeCost	DEL-TC-C1H10	ICR-TCvalue-C1H10	DEC-TCvalue-C1H10	DEL-TC-C2H10	REP-TC-C1H10
ObservationVariables	DEL-OBV-C1H11	ICR-OBVrfs-C1H11	DEC-OBVrfs-C1H11	DEL-OBV-C2H11	REP-OBV- C1H11
TimeConstraint	DEL-TimC-C1H12	ICR-TimCvalue-C1H12	DEC-TimCvalue-C1H12	DEL-TimC-C2H12	REP-TimCvalue-C1H12
ResourceConstraint	DEL-ResC-C1H12	ICR-ResCvalue-C1H12	DEC-ResCvalue-C1H12	DEL-ResC-C2H12	REP-ResC-C1H12

表 19 含有关键字的 RUCM 语句的变异算子(74 种)

	NO	MORE	LESS	AS WELL AS	PART OF	REVERSE	OTHER THAN
RFS	DEL-RFS-{C2F3, C6F1}	ICR-RFSsi-{C1F5, C3F3}	DEC-RFSsi-{C1F5, C6F1}	ADD-RFSflow-{C3F2, C3F3}	DEL-RFSflow-C3F2	SWAP-RFS-C1F5	REP-RFS-C3F3
RESUME STEP	DEL-RES-C2F5	ICR-RESsi-C1F6	DEC-RESsi-C1F6	-	-	SWAP-RES-C1F6	REP-RES-C1F6 REP-toABT-C6F3
ABORT	DEL-ABT-C2F6	-	-	-	-	-	REP-toRES-C3F2
IF-ELSE-ENDIF	DEL-IFELSE-C2F7	-	-	ADD-IFELSEcs-{C1F7, C6F1, C8F1} ADD-IFELSEas-C9F2	DEL-IFELSEcs-C1F7 DEL-IFELSEas-{C1F1, C1F2 C2F2, C3F1, C3F2 C3F5, C6F1, C6F3, C8F1}	SWAP-IFELSE-{C1F1, C6F1}	REP-IFELSE-{C1F1, C6F1}
DO-UNTIL	DEL-DO-C2F7	-	-	ADD-DOcs-{C1F7, C6F1, C8F1} ADD-DOas-C9F2	DEL-DOcs-C1F7 DEL-DOas-{C1F1, C1F2 C2F2, C3F1, C3F2 C3F5, C6F1, C6F3, C8F1}	-	REP-DO-{C1F1, C6F1}
VALIDATES THAT	DEL-VLD-C1F5	-	-	-	-	REP-VLD-{C6F1, C8F1}	REP-VLD-{C6F1, C8F1}
MEANWHILE	DEL-MW-{C3F1, C3F2}	-	-	-	-	SWAP-MW-{C1F1, C1F2, C3F1, C3F2, C6F1, C8F1}	REP-MW-{C1F1, C1F2, C3F1, C3F2, C6F1, C8F1}

4.2.4 缺陷生成策略的辅助规则

在变异分析的实践中，如何有效地控制‘变种(mutant)’的数量是一个非常重要的环节[114]。如文献[44]中所论述的，通常采用四种不同的技术对生成的变种进行挑选和删减：变种采样(*Mutant Sampling*)、变种聚类(*Mutant Clustering*)、筛选变异(*Selective Mutation*)、排序变异(*Higher Order Mutation*)。变种采样(*Mutant Sampling*)：随机挑选一个小的变种集合。变种聚类(*Mutant Clustering*)：使用聚类算法进行变种挑选。筛选变异(*Selective Mutation*)：基于特定的筛选规则选择一部分变异算子从而达到变种删减的目标。排序变异(*Higher Order Mutation*)：主要是识别那些不经常使用但是非常有用的变种。受这些前沿技术研究的启发，本文给出一系列指导规则用以创建成本-效益的缺陷生成策略。

Guideline 1: RUCM 缺陷在不同的环境下可能会有不同的重要性。例如，从系统开发者的角度看，C1F5 (用况规约中不正确的 RUCM 备选流分支)可能要比 C1R5 (用况与参与者之间不正确的关联关系)严重，然而对于需求人员，特别是在需求获取规约阶段，严重性则可能相反。同样，在某些特定环境下，*不完整性*(*Incompleteness*)可能要比*不可修改性*(*Unmodifiability*)更重要。一个 RUCM 缺陷的”重要性”的特性(即对应的一个 RUCM 变异算子)，在 RUCM 变异分析中，可以被用来实现一种筛选变异(*Selective Mutation*)的策略。

Guideline 2: 每一个 RUCM 变异算子只能跟一个具体的 RUCM 缺陷类别相对应，也就是说一个 RUCM 变种对应一个具体的 RUCM 缺陷分类。就 RUCM 变异算子而言，它是不会创建等价变种[44](*Equivalent Mutants*)的，即那些在结构上与原 RUCM 模型不一样但是在语义上它们是等同的变种。这是因为 RUCM 模型(用况图和用况规约)与程序代码不同，无论是对用况图还是对用况规约的修改一定会引起相应的结构或语义上改变。

Guideline 3: 某些 RUCM 缺陷只能进行手工创建而且使用 RUCM 变异算子生成 RUCM 缺陷的精力成本可能不同。例如，使用变异算子‘ADD-AF-C9F1’要比变异算子‘DEL-SAR-C2H2’费力，因为前者需要往用况规约中插入一个完整的 RUCM 备选流而后者只需要从用况规约中删除一个辅助参与者(Secondary Actor)。因此，根据 RUCM 变异算子定义一个成本-效益的故障注入策略是非常关键的。给定一个 RUCM 模型，其中成本(Cost)就是使用变异算子生成 RUCM 缺陷而作出的努力，效益(effectiveness)就是杀死相应变种的能力。成本(Cost)可以通过使用不同的变异删减技术(如 *Selective Mutation*)进

行降低。

Guideline 4: 一个 RUCM 缺陷，可能会有多种不同的实现，即不同的 RUCM 变异算子应用于不同的 RUCM 模型元素可能会产生同一种 RUCM 缺陷。例如，C1F3(不正确的 RUCM Step 排序)可以通过变异算子 SWAP-SenBF-C1F3(交换 RUCM 基本流中的两个 RUCM Step 的顺序)实现，也可以通过变异算子 SWAP-SenAF-C1F3(交换一个 RUCM 备选流中的两个 RUCM Step 的顺序)实现。因此，需要设计一种缺陷生成策略用以消除那些冗余的 RUCM 变种。冗余变种[120] (*Redundant Mutants*)：如果它们的输出结果与其他的变种是一样的，或者它们的输出结果可以通过其他变种的结果进行创建[120]。针对 C1F3，一种缺陷生成策略可以是仅使用 SWAP-SenBF-C1F3 或 SWAP-SenAF-C1F3 其一。

Guideline 5: RUCM 变异算子的创建机制(Section 4.2.3.2)，基于 9 种缺陷类别，可以辅助 RUCM 变异算子的挑选。所有的 RUCM 变异算子可以自然的划分为 9 个不同的类别，每一类都对应 RUCM 缺陷分类中的一种。依据当前要评审的问题的不同，研究者可以选择相应种类的 RUCM 变异算子。此外，还可以依据 RUCM 模型元素的不同选择相应的 RUCM 变异算子，因为每个 RUCM 元素都有可适用的变异算子。例如，假设一个评审方法只是针对用况规约进行设计的，那么那些为用况图设计的变异算子就不应该被选用。

4.3 案例研究

该小节对创建的 RUCM 变异算子设计相应的实验进行验证： 4.3.1 节描述了选取的案例，4.3.2 节介绍了实验的研究目标，4.3.3 节陈述了实验中设计的研究问题，4.3.4 节对实验的执行进行报告，4.3.5 节讨论实验数据和相关分析，4.3.6 节进行了相关的讨论。

4.3.1 案例描述

为了对 MuRUCM 方法进行验证，本章选取了 2 个工业案例和 9 个文献案例进行实验。表 20 描述了这些案例的基本特性，从中可以发现该实验中总共对 59 个用况进行变异分析(每个用况有且只有一个基本流(Basic Flow))。

4.3.1.1 工业案例

该实验中选用的第一个工业案例是航空领域的飞行控制系统(NAS)[102]，该案例的详

细描述可以参阅第 2 章 2.3.2 节。在该实验中选用以下 11 个用况：启动系统 Start System (UC1)，用于完成系统启动时的初始化相关操作；上电自检 Power-up Built-in Test (UC2)，主要负责系统上电后的一系列设备、部件检查工作；故障处理 Handle Faults (UC3) 主要被设计来解决系统运行中出现的各种错误和异常；数据采集 Sample Data (UC4)负责从各个传感器采集各种飞行数据；系统同步 Synchronize with SystemB (UC5) 负责实现与另一个冗余系统的同步工作。NAS 在设计上采用双机冗余的策略以保证其安全可靠；表决输入数据 Vote InputData (UC6)对收到的传感器采集数据进行验证和处理；表决输出数据 Vote OutputData (UC7)对计算后产生的飞行指令数据进行校验和处理；传输数据 Transmit Data (UC8)将各种表决后的数据传输给对方冗余系统；计算控制律 Calculate Control Law (UC9)，依据控制律计算公式使用表决处理后的飞行数据进行计算；输出飞行指令(UC10)，将表决后的飞行参数发送给舵机作动器；关闭系统 Shut Down (UC11)，系统掉电后的一系列数据回写等操作。

第二个工业案例也是来自航空工业领域，它是一个分区操作系统(OS)[121]。OS 主要为部署在各个分区中的不同应用提供可靠、安全的分区内或分区间的系统服务。该实验中选用如下 12 个用况进行建模和变异分析：*UC001*：提供以线程为主体的短 IPC(Inter-process Communication)消息发送功能。*UC002*：以线程为主体的短 IPC 消息接收功能。*UC003*：以线程为主体的短 IPC 消息发送后接收的复合功能。*UC004*：以线程为主体的完全 IPC 消息发送功能。*UC005*：以线程为主体的完全 IPC 消息接收功能。*UC006*：以线程为主体的扩展 IPC 消息发送功能。*UC007*：以线程为主体的扩展 IPC 消息接收功能。*UC008*：提供线程创建的系统功能，并限制该功能只能由特权线程调用。*UC009*：提供线程销毁的系统功能，并限制该功能只能由特权线程调用。*UC010*：提供线程启动的系统功能，并限制该功能只能由特权线程调用。*UC011*：提供线程停止的系统功能，并限制该功能只能由特权线程调用。*UC012*：提供线程悬挂的系统功能，并限制该功能只能由特权线程调用。

4.3.1.2 文献案例

该实验中选用了 9 个文献案例：ATM (Banking System)[122]、CMS (Crisis Management System)[123]、CPD (Car Part Dealer System)、 VS (Video Store systm)、CDS (Cab Dispatching system)、 PAY (Payroll System)、 OPS (Order Processing System)[28]、 SH (SafeHome

Project)[124]、ARENA[125]。ATM 是出现在文献[126]中的一个银行系统，Capozucca[127]等人定义了有关汽车碰撞的危机管理系统软件产品线的一系列需求，即 bCMS-SPL，本实验中选取了其中的一个用况 (Communicate with other coordinator)对其使用 RUCM 进行描述。CPD、VS、CDS、PAY 在文献[8,23]中用于对 RUCM 方法及其实现的从 RUCM 用况模型到 UML 分析模型的模型转换进行评估验证。OPS 选自教科书[28]，在该实验中选用了五个用况：*Fill and Ship Order*、*Give Product Information*、*Place Order*、*Update Account*、*Update Product Quantities*。SH 选自软件工程书籍[124]，实验中选用了三个用况：*Verify Account*、*Start Monitoring Window and Door*、*Review CCTV Data*。ARENA 来自面向对象建模教程[125]，该实验选用了一个用况：*Announce Tournament*。

表 20 实验案例的特性描述

Model	Case Study											Total
	NAS	OS	ATM	CMS	CPD	CDS	PAY	VS	OPS	SH	ARENA	
#Actor	7	13	2	2	4	8	3	4	5	5	4	57
#Include	4	1	3	0	2	5	0	4	4	2	0	25
#Extend	2	0	0	0	0	0	0	0	0	0	0	2
#Generalization	2	0	0	0	0	0	0	0	0	0	0	2
#Association	7	13	4	2	4	8	3	6	7	6	4	64
#Basic Flow	11	12	4	1	6	7	1	8	5	3	1	59
#Alternative	28	30	7	7	12	12	3	12	6	8	6	131
#RFS	28	33	7	7	12	12	3	12	6	10	9	139
#RESUME	20	2	1	7	5	7	4	8	1	1	5	61
#ABORT	8	28	7	1	8	5	0	9	5	5	1	77
#IF-ELSE-	23	10	3	12	8	2	6	12	1	4	0	81
#DO-UNTIL	2	0	0	3	6	3	1	0	4	1	0	20
#VALIDATES	36	30	10	1	5	11	3	14	5	12	9	136
#MEANWHILE	8	0	1	1	1	2	2	0	1	0	2	18
#Precondition	11	12	4	1	6	7	1	8	5	3	1	59
#Post-condition	39	42	11	8	18	19	4	20	11	11	7	190

4.3.2 实验目标

本文研究中的目标之一是支持基于 RUCM 用况的需求评审。为了实现此目标，本章提出了 MuRUCM 的方法，从而可以系统化的为用况模型注入各种不同类型的需求缺陷，为需求评审准备评审源，也为客观公正的对各种评审技术的评估提供准备。本节采用 *G-Q-M*(Goal-Question-Metric)[128]方法对实验目标进行如下的形式化阐述。

Goal 1。从***支撑需求评审(即是否涵盖文献中常见的缺陷类型)***的视角出发，以***评估***为目标，针对***其完整性***，在***学术实验室***的环境下，对 ***RUCM 缺陷分类***进行分析。

Goal 2。从***支撑需求评审(即创建不同类型的 RUCM 需求缺陷)***的视角出发，以***评估***为目标，针对***其有效性***，在***学术实验室***的环境下，对 ***RUCM 变异算子***进行分析。

Goal 3。从***支撑需求评审(即辅助生成不同类型的 RUCM 需求缺陷)***的视角出发，以***评估***为目标，针对***其有效性***，在***学术实验室***的环境下，对 ***RUCM 缺陷生成策略的辅助规则***进行分析。

Goal 1 通过与文献研究进行对比从而确定 RUCM 缺陷分类是否完整，即它能否涵盖文献中常见的用况需求缺陷。***Goal 2*** 和 ***Goal 3*** 分别研究 RUCM 变异算子和 RUCM 缺陷生成策略辅助规则的有效性。 为了系统化的研究以上实验目标，本章设计了以下的研究问题(4.3.3 节)。

4.3.3 研究问题

为了对 MuRUCM 进行系统化的分析，本节基于 4.3.2 节中的研究目标设计以下的研究问题：

RQ1：与文献研究相对比，RUCM 缺陷分类是否完整？该研究问题主要回答 RUCM 缺陷能否涵盖文献研究中常见的用况缺陷类型，它是对研究目标 ***Goal 1*** 的细化。

为了对研究目标 ***Goal 2*** 进行全面分析，设计了研究问题 RQ2-RQ4 。

RQ2：所有的 RUCM 变异算子能否使用？该研究问题旨在研究 RUCM 变异算子的可用性。

RQ3：变异算子生成的 RUCM 变种是否有用？为了回答该问题，本章用生成的 RUCM 变种来评估 3 种不同的 RUCM 用况场景生成策略，并以变异数(Mutation Score)为平均指标。

RQ4：RUCM 变异算子的分布是怎样的？即实验中使用它们创建相应的 RUCM 缺陷实例的分布。

RQ5：RUCM 缺陷生成策略的辅助规则是否有用？该研究问题直接来自研究目标 ***Goal 3***。

4.3.4 实验执行

为了回答 4.3.3 节中的研究问题，该实验由以下两个具体活动构成：创建用况变种(4.3.4.1 节)、执行变异分析(4.3.4.2 节).

4.3.4.1 创建 RUCM 变种

为了系统化的创建 RUCM 变种，本实验依据节阐述的 MuRUCM 规则，进行如下的选择设计。1)，该实验通过使用相应的 RUCM 变异算子创建所有的 RUCM 缺陷类型(*C1-C9*)(即 Guideline 5)。2)，由于并不知道各个缺陷类型在实际工程项目中的重要程度(即工业案例的开发历史记录不可用)，该实验并不区分不同 RUCM 缺陷类型的重要性(Guideline 1)。例如，给定一个 RUCM 用况规约，如果 *solA* 识别出 3 个 *C1* 类型的缺陷实例，而 *solB* 识别出 1 个 *C1* 类型缺陷实例和 2 个 *C6* 类型缺陷实例，那么 *solA* 和 *solB* 的效能是一样的，因为它们取得一样的 MS。3)，针对那些可以使用不同的 RUCM 变异算子生成的 RUCM 缺陷类型，依据 Guideline 3 和 Guideline，该实验选择那些建模成本小的 RUCM 变异算子。4)，依据 Guideline 2，每一个 RUCM 变异算子对应一个特定的 RUCM 缺陷类型，该实验中不需要考虑等效变种的影响。

针对用况图，进行如下的 RUCM 变种生成。针对一个参与者(Actor)：修改它的名字，或者将它的名字替换为另一个 Actor 的名字。 参与者与用况之间的关联关系： 删除相应的关联关系，或者变换关联端参与者(或用况)为另外的一个参与者(或用况)。两个参与者之间的泛化关系：删除该泛化关系，或者变换泛化关系的方向(即互换基类参与者与子类参与者之间的角色)。用况之间的«include»或«extend»关系：删除该«include»（或«extend»）关系，或者变换«include»（或«extend»两端的用况的角色。

针对用况规约，主要从以下几个方面创建相应的 RUCM 变种生成。对每一个 RUCM 备选流：修改相应的分支语句，即 RFS-FlowName-Step。针对关键字‘RESUME STEP’：修改相应的返回语句 step。针对 RUCM 条件语句(如，‘IF-THEN-ELSE-ELSEIF-ENDIF’、

‘DO-UNTIL’)，修改相应的条件描述语句(Condition Step)。针对 RUCM 事件流中的动作语句(Action Step)：进行删除一个或多个语句、添加一个或多个语句、将其中的几个语句进行互换的操作。针对用况前置条件或后置条件：修改相应的语句描述。在进行如上的变种操作时，系统化的使用 RUCM 变异算子。

4.3.4.2 执行变异分析

意识到开展需求评审的困难，例如，如何控制评审人员的个人经验、能力的差异对评审结果的影响等，本章的实验中采用故障注入这种比较客观的方式替代传统的手工需求评审的过程对 MuRUCM 进行评估。也就是说，首先系统化的手动创建 RUCM 变种，然后根据不同的 RUCM 覆盖标准生成相应的 RUCM 用况场景，最后针对每一组用况场景计算它们各自的变异分数(MS)，其中 MS 定义如下：

$MS = (the\ number\ of\ killed\ mutants)\ /(total\ number\ of\ seeded\ mutants)$。

具体讲，给定一个用况规约变种，该实验采用三种不同的 RUCM 覆盖标准：the *All Condition* coverage、*All FlowOfEvents* coverage 和 *All Sentence* coverage[9]生成用况场景。相应的获得三组用况场景：$S_{All-Condition}$、$S_{All-FlowOfEvents}$、$S_{All-Sentence}$。针对每一组用况场景，计算它的变异分数。例如，$MS_{S_{All-Condition}}$对应着 *All Condition* coverage。最后，使用$MS_{S_{All-Condition}}$、$MS_{S_{All-FlowOfEvents}}$、$MS_{S_{All-Sentence}}$作为评价指标对不同的 RUCM 覆盖策略进行评估，即判定哪种标准可以发现更多的需求缺陷。

4.3.5 结果分析

本小节针对每个研究问题报告相应的实验结果并进行分析。

4.3.5.1 RQ1 的结果与分析

RQ1 针对 RUCM 缺陷分类的完整性进行评估。为了回答这个问题将 RUCM 需求缺陷分类与文献[6,117,115,116]中提出的四种缺陷分类进行对比。结果显示 RUCM 缺陷分类涵盖了[6,117,115,116]中提出的缺陷分类。RUCM 缺陷分类适应于整个用况模型包括用况图和用况规约。此外，与文献[6,117,115,116]相比，RUCM 缺陷分类不仅定义了更多的缺陷类型而且给出了更加详细明确的缺陷定义。尽管 Anda 和 Sjøberg[115]给出的缺陷分类也适用于整个用况模型，但是 RUCM 缺陷分类定义了更多的缺陷类型(例如

Intestability)，而且每种缺陷类型都有一个针对 RUCM 模型元素的明确定义。更进一步，RUCM 缺陷分类是系统化依据工业标准 IEEE Std. 830-1998[72]定义的。尽管文献[116]也是依据标准进行的定义，但是 RUCM 缺陷分类针对每一种缺陷给出了更加明确的定义，它也更加适合对不同的需求评审技术进行评估。

4.3.5.2 RQ2 的结果与分析

本实验中针对选用的 11 个案例系统化的创建相应的用况变种，表 21 展示了所有的变种信息，从中可以发现总共创建了 6508 个用况变种。如表 21 所示，一些缺陷类型可以使用不同的变异算子实现，一些缺陷类型可以使用特定的一种变异算子实现，一些缺陷类型必须要使用多个变异算子才能实现。例如，如表 21 所示，变异算子 `REP-ARN-C1AR1` 和缺陷类型 `C1AR1` 之间的映射关系就是一对一的，在该实验中变异算子 `REP-ARN-C1AR1` 使用了 36 次创建了 36 个变种，相应的产生了 36 `C1AR1` 缺陷实例。针对缺陷类型 `C1F3` 而言，它可以使用变异算子 `SWAP-SenBF-C1F3` 或 `SWAP-SenAF-C1F3` 进行创建，其中变异算子 `SWAP-SenBF-C1F3` 使用了 148 次，`SWAP-SenAF-C1F3` 使用了 268 次，如表 21 所示。

对于特殊的缺陷类型，如 `C7UCM1`，需要使用多个变异算子才能实现。在该实验中，如表 21 所示，变异算子 `ADD-SenBF-C9F2` 和 `DEL-SenAF-C2F2` 用来创建 `C7UCS1` 缺陷实例。

对于其他的六缺陷类型(即，`C2UCS1`、`C3UCS1`、`C3UCS2`、`C3UCS3`、`C3UCS4`、`C3UCM1`)，可以使用不同的变异算子进行实现。例如，实验中采用 `DEL-PAR-C2H1` 创建 `C3UCM1` 实例。

最终在该实验中，通过使用 219 种 RUCM 变异算子针对 104 种缺陷类型创建了 6508 个用况变种，使得每种缺陷类型都被至少覆盖到一次。实验数据和实验经验显示 RUCM 用况变异算子是易于使用的。

表 21 实验中生成的用况变种及使用的相应变异算子(RQ2)

Mutation Operator	#Mutants	Mutation Operator	#Mutants	Mutation Operator	#Mutants	Mutation Operator	#Mutants
REP-ARN-C1AR1	36	DEL-AR-C2AR1	32	REP-ARN-C3H1	38	ADD-SenPreC-C6H1	26
REP-UCN-C1UC1	47	DEL-UC-C2UC1	41	REP-BD-C3H2	45	DEL-SenPreC-C6H1	31
SWAP-GAR-C1R1	4	DEL-AR-C2R1	4	REP-PreC-C3H3	45	REP-PreC-C6H1	6
REP-GAR-C1R1	11	DEL-UC-C2R2	34	DEL-SenBF-C3F1	18	DEL-SenBF-C6F1	47
SWAP-GUC-C1R2	13	DEL-INC-C2R3	21	SWAP-SenBF-C3F1	6	SWAP-SenBF-C6F1	12
REP-GUC-C1R2	12	DEL-EXD-C2R4	4	REP-BF-C3F1	4	DEL-SenAF-C6F1	23
SWAP-INC-C1R3	16	DEL-ASSO-C2R5	37	DEL-IFELSEas-C3F1	4	SWAP-SenAF-C6F1	9
REP-INC-C1R3	24	DEL-PAR-C2H1	16	SWAP-IFELSE-C3F1	2	REP-AS-C6F1	3
SWAP-EXD-C1R4	11	DEL-SAR-C2H2	23	REP-IFELSE-C3F1	2	DEL-RFS-C6F1	12
REP-EXD-C1R4	14	DEL-BD-C2H3	42	DEL-DOas-C3F1	7	DEC-RFSsi-C6F1	13
REP-ASSO-C1R5	16	DEL-PreC-C2H4	44	REP-DO-C3F1	5	ADD-IFELSEcs-C6F1	3
SWAP-AR-C1H1	7	DEL-TimC-C2H12	21	SWAP-MW-C3F1	4	DEL-IFELSEas-C6F1	22
REP-AR-C1H1	13	DEL-EXD-C2H6	5	REP-MW-C3F1	2	SWAP-IFELSE-C6F1	3
SWAP-AR-C1H2	9	DEL-GUC-C2H7	5	DEL-MW- C3F1	4	REP-IFELSE-C6F1	3
REP-AR-C1H2	16	DEL-AF-C2F1	130	SWAP-AF-C3F2	4	ADD-DOcs-C6F1	3
REP-SenBD-C1H3	22	DEL-AS-C2F2	91	DEL-SenAF-C3F1	29	DEL-DOas-C6F1	14
DEL-SenBD-C1H3	18	DEL-SenBF-C2F2	39	SWAP-SenAF-C3F2	7	REP-DO-C6F1	3
REP-BD-C1H3	14	DEL-SenAF-C2F2	89	REP-AF-C3F2	4	REP-VLD-C6F1	3
DEL-SenPreC-C1H4	32	DEL-IFELSEas-C2F2	37	ADD-RFSflow-C3F2	4	SWAP-MW-C6F1	3
REP-PreC-C1H4	18	DEL-DOas-C2F2	12	DEL-RFSflow-C3F2	16	REP-MW-C6F1	3
REP-INC-C1H5	20	DEL-RFS-C2F3	86	REP-toRES-C3F2	3	ADD-SenPostC-C6F2	101
REP-EXD-C1H6	6	DEL-PostC-C2F4	127	DEL-IFELSEas-C3F2	19	REP-SenPostC-C6F2	27
REP-GUC-C1H7	6	DEL-RES-C2F5	57	DEL-DOas-C3F2	16	REP-PostC-C6F2	12
DEL-SenBF-C1F1	21	DEL-ABT-C2F6	41	SWAP-MW-C3F2	3	DEL-SenBF-C6F3	26
SWAP-SenBF-C1F1	6	DEL-IFELSE-C2F7	69	REP-MW-C3F2	1	SWAP-SenBF-C6F3	7
REP-BF-C1F1	4	DEL-DO-C2F7	17	DEL-MW-C3F2	3	DEL-SenAF-C6F3	11
DEL-IFELSEas-C1F1	4	**C2UC1:** DEL-UC	42	ICR-RFSsi-C3F3	27	SWAP-SenAF-C6F3	6
SWAP-IFELSE-C1F1	3	**Total in C2**	**1166**	ADD-RFSflow-C3F3	14	REP-AS-C6F3	4
REP-IFELSE-C1F1	3	REP-UCN-C4UC1	51	REP-RFS-C3F3	5	REP-toABT-C6F3	7
DEL-RCE-C1H8	7	REP-BD-C4H1	130	REP-PostC-C3F4	130	DEL-IFELSEas-C6F3	12
REP-DO-C1F1	3	REP-PreC-C4H2	130	DEL-SenBF-C3F5	8	DEL-DOas-C6F3	12
SWAP-MW-C1F1	3	REP-AS-C4F1	84	DEL-SenAF-C3F5	8	**Total in C6**	**467**
REP-MW-C1F1	4	REP-AS-C4F2	169	DEL-IFELSEas-C3F5	8	DEL-SenBF-C8F1	29
DEL-SenAF-C1F2	96	REP-PostC-C4F3	125	DEL-DOas-C3F5	26	SWAP-SenBF-C8F1	6
SWAP-SenAF-C1F2	89	REP-AS-C4UCS1	125	***C3UCS1:*** REP-UCN-C1UC1	42	DEL-SenAF-C8F1	11
DEL-IFELSEas-C1F2	73	REP-AS-C4UCS2	124	***C3UCS2:*** REP-AS-C4F1	131	SWAP-SenAF-C8F1	11
SWAP-MW-C1F2	6	**Total in C4**	**938**	***C3UCS4:*** REP-AS-C4F1	29	ADD-IFELSEcs-C8F1	21
DEC-TCvalue-C1H10	9	REP-ARN-C5AR1	38	***C3UCM1:*** DEL-PAR-C2H1	42	DEL-IFELSEcs-C8F1	14
SWAP-SenBF-C1F3	148	REP-UCN-C5UC1	51	**Total in C3**	**765**	ADD-DOcs-C8F1	9
SWAP-SenAF-C1F3	268	REP-BD-C5H1	51	**C7UCS1:** ADD-SenBF-C9F2, DEL-SenAF-C2F2	101	DEL-DOcs-C8F1	9
DEL-SenPostC-C1F4	29	REP-PreC-C5H2	51			REP-VLD-C8F1	12
REP-PostC-C1F4	15	REP-AS-C5F1	247	**C7UCM1:** ADD-SenBF-C9F2, DEL-SenAF-C2F2, DEL-SenBF-C9F2	31	SWAP-MW-C8F1	2
DEC-TimCvalue-C1H12	24	REP-PostC-C5F2	131			REP-MW-C8F1	4
ICR-RFSsi-C1F5	51	**Total in C5**	**569**	**Total in C7**	**132**	**Total in C8**	**128**
DEC-RFSsi-C1F5	34	DEL-IFELSEcs-C1F7	38	ADD-AR-C9AR1	41	ADD-INC-C9H4	16
SWAP-RFS-C1F5	12	ADD-DOcs-C1F7	17	ADD-UC-C9UC1	9	ADD-EXD-C9H5	9
DEL-VLD-C1F5	21	DEL-DOcs-C1F7	17	ADD-AR-C9R1	11	ADD-GUC-C9H6	14
ICR-RESsi-C1F6	28	ADD-IFELSEcs-C1F7	62	ADD-UC-C9R2	14	ADD-AF-C9F1	77
DEC-RESsi-C1F6	22	ADD-SenBD-C9H2	42	ADD-INC-C9R3	8	ADD-SenBF-C9F2	58
SWAP-RES-C1F6	6	ADD-SenPreC-C9H3	131	ADD-EXD-C9R4	8	ADD-SenAF-C9F2	111
REP-RES-C1F6	7	ADD-SenPostC-C9F3	131	ADD-ASSO-C9R5	35	ADD-IFELSEas-C9F2	71
Total in C1	**1515**	ADD-DOas-C9F2	21	ADD-SAR-C9H1	21	**Total in C9**	**828**

4.3.5.3 RQ3 的结果与分析

为了回答 RQ3，使用 RQ2 中针对 11 个案例创建的 6508 个变种对不同的 RUCM 覆盖策略的效能进行实验评估。如 4.3.4 节论述的，该实验中采用变异分数 *Mutation Score* (MS) 作为评价指标。

实验结果在图 29 进行展示。从中可以发现，基于评价指标变异分数(MS)，创建的用况变种可以很好的区分出三种不同的 RUCM 覆盖策略。针对 *C1* d 类型的缺陷，*All Condition* 覆盖标准取得最高的变异分数(0.96)，然后是 *All FlowsOfEvents* 覆盖标准(MS=0.82)，而 *All Sentences* 覆盖标准取得最低的变异分数(0.66)。对其他的缺陷类型(*C2-C8*)，可以发现同样的结果模式。这其中的原因是：与其他两种覆盖标准相比较，*All Condition* 覆盖标准关注引起用况事件流产生分支的每一个条件，同时又产生更多的用况场景。以上的实验数据明确显示，使用 RUCM 变异算子创建的用况变种能够非常有效的区分出不同的 RUCM 覆盖标准它们针对缺陷发现率(即 MS)的性能。

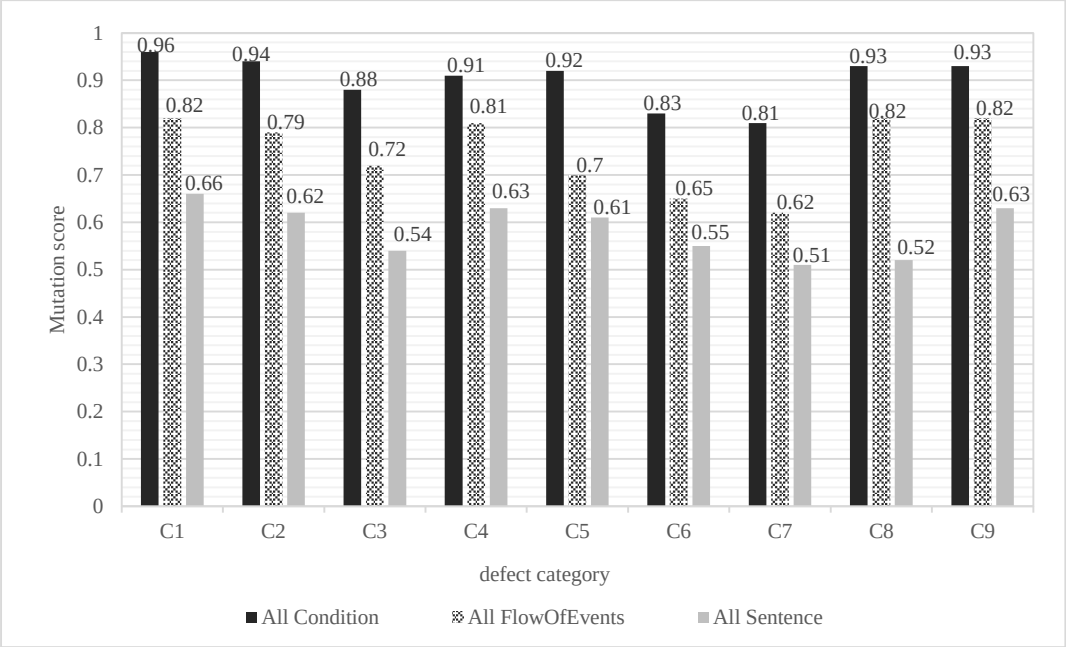

图 29 三种不同 RUCM 覆盖标准的相应变异分数(RQ3)

更进一步，如 4.3.4.2 节所论述的，给定一个 RUCM 用况规约，针对每一个 RUCM 覆盖标准(*All Condition* coverage criterion、*All FlowsOfEvents* coverage criterion、*All Sentences* coverage criterion)，可以分别计算得到$MS_{S_{All-Condition}}$、$MS_{S_{All-Condition}}$、$MS_{S_{All-Condition}}$。既然该实验中总共有 59 个用况，那么也就分别有相应的 59 组

$MS_{S_{All-Condition}}$、$MS_{S_{All-Condition}}$、$MS_{S_{All-Condition}}$，因此可以使用它们进行统计分析。本实验中采用 Vargha-Delaney 统计[77]、Wilcoxon 秩和检验[79]同时搭配 Bonferroni[129]修正，对这些计算后的变异分数进行统计分析。其中，Vargha-Delaney 的统计计算数值是 $\hat{A}_{12}$，它是一个效应评价因子。在该实验中，$\hat{A}_{12}$ 用来比较两组不同 RUCM 覆盖策略生成的用况场景 *A* 和 *B*，它们能够取得更高变异分数的概率。如果 $\hat{A}_{12}$=0.5，则 *A* 和 *B* 具有同样的性能；如果 $\hat{A}_{12}$>0.5，那么 *A* 有更高的机会发现更多的需求缺陷，反之 *B* 有更高的机会发现更多的需求缺陷。Wilcoxon 秩和检验搭配 Bonferroni 修正方法用来计算*p*-values，从而确定 *A* 和 *B* 之间是否存在显著差异，该实验设定置信水平是 0.05，即如果一个*p*-value 小于 0.05 那么就存在显著性的差异。相应的统计数据都报告在表 22 中。

表 22 Vargha-Delaney 统计, Wilcoxon 秩和检验搭配 Bonferroni 修正，显著水平为 0.05 (RQ3)

Pair of the RUCM coverage criteria (A vs. B)	**$\hat{A}_{12}$**	***p*-value**
All-Condition *vs.* All-FlowOfEvents	0.97	< 0.05
All-Condition *vs.* All-Sentence	1	< 0.05
All-FlowOfEvents *vs.* All-Sentence	0.96	< 0.05

基于表 22 中的统计结果可以得出结论：*All Condition* 覆盖标准要比 *All FlowsOfEvents* 覆盖标准显著得好($\hat{A}_{12}$=0.97 且*p*-value<0.05)，而 *All FlowsOfEvents* 覆盖标准要比 *All Sentences* 覆盖标准显著得好($\hat{A}_{12}$=0.96 且*p*-value<0.05)。

4.3.5.4 RQ4 的结果与分析

图 30 展示了实验中生成的用况变种的分布，从中可以发现 6508 个用况变种中包含：1515 个 *Incorrectness (C1)*类型的实例，1166 个 *Incompleteness (C2)* 类型的实例，765 个 *Inconsistency (C3)* 类型的实例，938 个 *Ambiguity (C4)* 类型的实例，569 个 *Incomprehensibility (C5)* 类型的实例，467 个 *Intestability (C6)* 类型的实例，132 个 *Unmodifiability (C7* 类型的实例，128 个 *Infeasibility (C8* 类型的实例，828 个 *Over-Specification (C9)* 类型的实例。

图 30 中的数据显示 *C1* (*Incorrectness*)和 *C2* (*Incompleteness*)相应的缺陷实例是 6508 个变种中最多的两类。这是因为这两类缺陷类型对应的变异算子非常易用。例如，通过

修改 RUCM 用况规约中的一个 RUCM 语句、一个事件流(flow)或者是一个关键字就可以创建一个 *C1* (*Incorrectness*)实例。对于 *C2* (*Incompleteness*)，通过删除 RUCM 用况规约中某个模型元素就可以创建一个 *C2* 的实例。在所有的 RUCM 缺陷分类中，*C7* (*Unmodifiability*)和 *C8* (*Infeasibility*)需要花费更多的建模精力，因为这两类缺陷类型的最终结果反应了需求管理和系统开发中的问题。具体地讲，一个典型的 *C7* (*Unmodifiability*)类型的缺陷实例就是没有使用合适的«Include»或«Extend»对用况进行组织管理，另一种情形就是在 RUCM 用况规约中没有使用合适的边界事件流(*Bounded Alternative Flow*)替代多个可以合并的特定事件流(*Specific Alternative Flow*)。为了创建 *C7* (*Unmodifiability*)类型的实例，需要将«Include»或«Extend»类型的用况插入到对应的关联端用况中。对于 *C8* (*Infeasibility*)类型的实例创建，针对不同的应用领域，需要需求人员要额外的具备相应的领域开发知识。例如，在进行通讯系统的需求开发过程中，当对一个通讯媒介进行需求建模时，数据传输的速率和数据精度都需要进行精确的描述，而这些信息的准确性最终都要依据开发者从系统实现的角度进行确认和验证。

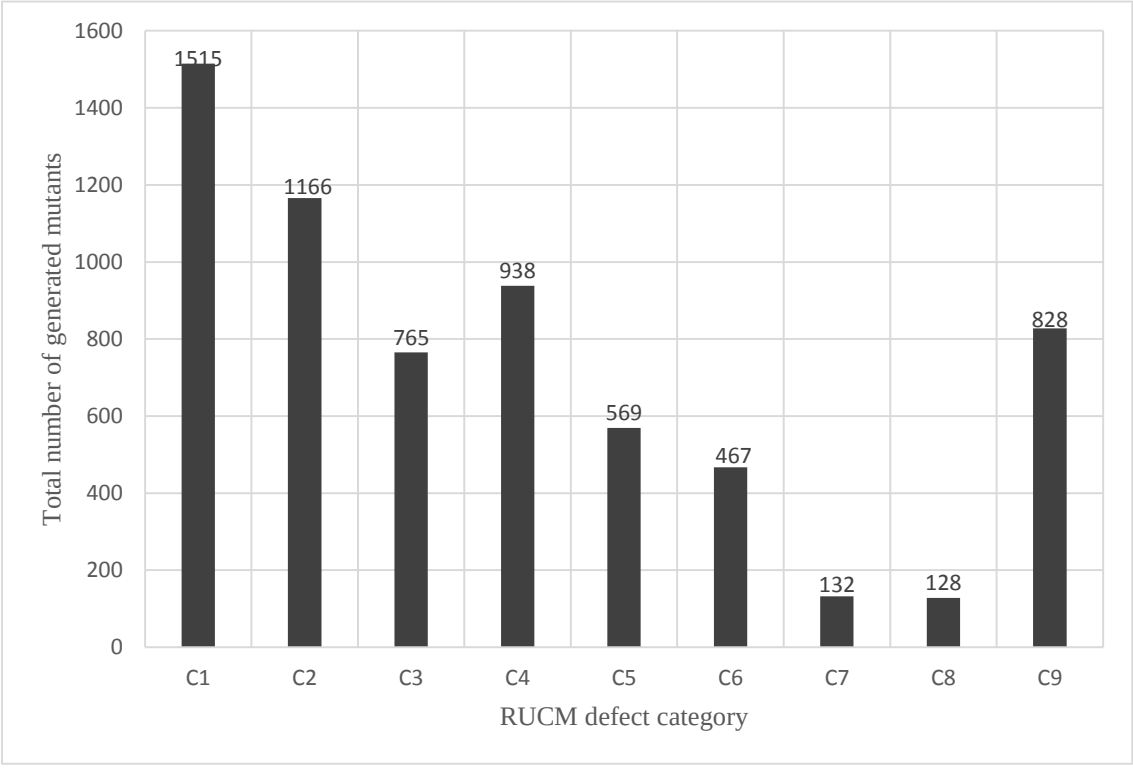

图 30 实验中生成的用况变种的分布(RQ4)

4.3.5.5 RQ5 的结果与分析

该研究问题旨在回答本章提出的缺陷生成策略的辅助规则是否有效。下面总结实验中使用它们的一些经验。

在本实验中，Guideline 1 和 Guideline 2 帮助定义确切的变异分数计算公式。具体讲，在该实验中并没有区分不同的缺陷类型的重要性(Guideline 1)，因为实验中并没有获取工业案例的需求开发的历史记录，因此不需要为每一种缺陷类型设定相应的权限值。另外根据 Guideline 2，RUCM 变异算子也不会引入等效变种，因此也不用考虑等效变种在设计变异分数计算公式的影响。基于以上事实，该实验中可以将变异分数计算公式：$\frac{\sum_{i=1}^{9}(\#\ of\ \ killed\ C_i\ mutants)*W_i}{\sum_{i=1}^{9}((\#\ of\ seeded\ C_i\ mutants)-(\#\ of\ C_i\ equivalent\ mutants))*W_i}$; $(\sum_{i=1}^{9} W_i) = 1$ 简化为如下的计算公式： $S = (\#\ of\ killed\ mutants)\ /(\#\ of\ seeded\ mutants)$。

Guideline 3 和 Guideline 4 主要考虑不同的变异算子在使用过程中的建模工作量。实验中可以发现这两个规则非常实用，因为它们总会尽可能的帮助挑选那些建模工作量小的变异算子。 例如，变异算子 `DEL-SenBF-C1F1` 和 `SWAP-SenBF-C1F3` 都可以生成 *C1(Incorrectness)*类型的实例，然而前者的建模工作量要小，因为它只需要从用况规约中删除一个 RUCM 语句，而不需要交互两个 RUCM 语句(`SWAP-SenBF-C1F3` 的实施)。

依据预先定义的缺陷类型，即不同类型的缺陷实例可以通过使用相应的变异算子生成， 在实验中只需要选择正确的变异算子进行使用即可。如 Guideline 5 所示，RUCM 变异算子是根据 9 种不同缺陷类型进行分类组织的。因此 ，给定要创建的缺陷类型，Guideline 5 有助于挑选相应的变异算子。 例如，如果实验设计中需要创建 *Incorrectness* 类型的缺陷实例，那么只需选择 *Incorrectness(C1)*对应的变异算子(如变异算子命名中包含’C1’的那些)。

4.3.6 实验讨论

DeMillo[130]等首先提出了变异分析技术，他们论述道 “构建一组完整的变异算子对于实时程序变异是至关重要的[130]”。本文认为采用严谨的方法系统化的创建相应的变异算子是实施任何变异分析的首要步骤，而这对于针对用况评审技术的基于故障注入方式的评估方法更加重要。

RUCM 变异算子是针对 RUCM 用况模型设计的，但是它们也可适用于其他的用况

模型。针对 RUCM 用况图的变异算子可以直接使用到其他的 UML 用况图模型中；针对 RUCM 用况规约的变异算子，可以不用进行显著的修改就能够适用于其他的用况规约，因为 RUCM 用况规约模板在设计时对当前有关用况规约模板的文献研究进行了系统化的调研并捕获了那些最常用的构建元素。

此外，本文提出的创建 RUCM 变异算子的 5 维机制是严格遵循工业标准 HAZOP[73] 和 IEEE Std. 830-1998[72]的，这种机制也可以帮助其他研究学者针对特定的研究问题定义出相应变异算子。实验结果显示本章定义的 RUCM 变异算子可以应用到不同领域的不同复杂程度的用况模型，而相应的缺陷生成策略辅助规则的使用经验也非常振奋人心，本实验中成功的创建了 6508 个用况变种并覆盖了每一种缺陷类型(*C1-C9*)。

4.4 有效性风险

4.4.1.1 构建风险分析

一个影响该实证研究的有效性的风险是：采用 RUCM 变异技术进行手工的故障注入时可能会引入额外的人为的需求缺陷。在具体的实验过程中，针对工业案例的研究，一位来自工业合作方的负责相关案例的工程师协助完成相关的实验。经过具体的培训，该工程师创建了工业案例的初始的 RUCM 模型，然后作者从遵循 RUCM 限制规则的角度出发，对创建的 RUCM 模型进行修正，并且每次修正都得到工业工程师的确认。因此，可以认为创建的 RUCM 模型并不存在额外未知的需求缺陷。针对其他的文献案例，作者非常仔细的对其采用 RUCM 进行描述，而未改变其原始的语义。这些文献案例早已用作 RUCM[8,23]的评估案例。此外，在进行故障注入时，作者严格遵循 RUCM 变异指导规则(4.2.4 节)从而最大程度的避免额外需求缺陷的意外引入。

4.4.1.2 内部风险分析

该实验中采用了来自不同领域的 59 个用况，它们的复杂程度是不受控制的，因此无法实现对实验对象的随机选择。由于无法获取工业合作方的需求开发历史记录，因此也无法对注入的需求故障进行针对性的控制。

4.4.1.3 结论风险分析

为了评估 RUCM 变异算子的有效性，即能否有效的区分三种不同的 RUCM 覆盖策

略：*All Condition Coverage*、*All FlowOfEvents Coverage*、*All Sentence Coverage*。实验中采用了变异分数(Mutation Score(MS))作为有效性平均指标。具体讲，首先用选取的 59 个用况创建用况变种完成故障注入，然后对每一个用况变种分别采用三种不同的 RUCM 覆盖策略生成相应的用况场景，最后计算每一组用况场景的相应 MS 值。由于实验中选用了 59 个用况，因此针对每一个 RUCM 覆盖策略就会有 59 个 MS 值，这样就可以用它们进行统计分析。该实验遵循严格的统计分析过程对实验结果数据进行统计分析。Vargha-Delaney 统计方法、Wilcoxon 秩和检验同时搭配 Bonferroni 修正方法被用来分析数据，其中 Bonferroni 修正用法用于对多重比较中 Wilcoxon 秩和检验返回的 p-value 进行修正以防止第 I 类统计错误。另外，根据文献[131]的指导，Vargha-Delaney 统计的$\hat{A}_{12}$效应因子与 Wilcoxon 秩和检验一起来对统计结果进行更好的解释说明。

4.4.1.4 外部风险分析

有关软件工程实验的一个通常的外部风险就是：实验结论的通用性。该实验中采用了两个工业案例[102]，人们可能会争论说其实验结果不适用于其他的案例。然而，这样的外部风险是进行实证研究非常普遍的。此外，除了工业案例，该实验还采用了 9 个不同的文献案例用以对 RUCM 变异算子进行成本-效益分析。在实验中，无论是文献案例还是工业案例，它们都显示一致的结论。

4.5 相关研究工作的对比

4.5.1 变异分析/变异测试

变异分析又叫做变异测试是被设计来对测试用例的有效性进行评估的一种技术[43]。变异分析被广泛地应用于各种编程语言的源程序中(即程序变异[44])如 C[45]、Java[46]、Ada[47]。变异分析也被应用到程序规约说明中，即规约说明变异(*Specification Mutation*)[44]，例如，在文献[48]中，作者使用变异分析技术对有限状态机进行验证。文献[44]是针对变异分析的最新文献综述，文献[44]的研究显示研究者对于程序变异的研究要比对规约说明变异的研究多。变异分析也已引入进了模型驱动的软件工程中[49,50]。然而，目前的研究文献中还没有针对用况模型的变异分析技术。

针对软件程序的传统的变异分析的过程如[44]所述：给定一个程序p，一组相应的问

题程序 p'(叫做变种)是基于预定义的变异算子生成的。然后使用生成的p'执行一个测试套件T，其中测试套件T在使用源程序p时是正确、成功执行的。如果测试套件T执行p'的结果与执行p的结果不一致，相应的变种就被‘杀死’。变异分数(全部杀死的变种与创建的全部变种的比值[44])就被计算来评估测试套件T的有效性。

受传统的变异分析技术的启发，本文将变异分析应用到用况模型中，从而系统化的创建不同的用况缺陷，最终支持对不同的基于 RUCM 的用况评审技术进行公平有效的评估。在本文的语境下，一个变种就是一个包含特定需求缺陷的用况模型。而不同的基于 RUCM 的用况评审技术所扮演的角色就如同变异测试中的测试套件。

4.5.2 用况模型的需求缺陷分类

Anda 和 Sjøberg[115]提出一种基于检查列表的阅读技术用于检查用况模型中的缺陷。他们针对用况模型给出一种缺陷分类的方案：缺少(*Omission*)、不正确(*Incorrect Fact*)、不一致(*Inconsistency*)、模糊(*Ambiguity*)、额外信息(*Extraneous Information*)。通过将这些缺陷类型指标应用到用况模型的不同元素从而创建相应的用况缺陷类型。文献[6]给出了一种缺陷分类，该缺陷分类主要针对用况规约中的语义错误，随后 Phalp[117]等人基于文本理解的相关理论对这种缺陷类型进行了细化，如同文献[6]所述，该缺陷分类只针对用况规约而不能涵盖整个用况模型。在文献[116]中，作者依据 IEEE Std. 830-1998[72]定义了一种缺陷分类用来进行需求评审，并给出了一组用况质量评价指标：一致性(*consistency*)、完整性(*completeness*)、正确性(*correctness*)、不模糊(*unambiguity*)、可验证(*verifiability*)、可修改(*changeability*)、可追踪(*traceability*)、优先化(*prioritization*)。

综上所述，Denger[116]等人遵循标准 IEEE Std. 830-1998[72]给出了缺陷分类，但是其中的分类定义比较抽象概括，研究者需要对其进行细化才可以使用，不容易直接应用于工业实践中。因此，可以把这种分类看作是定义 RUCM 缺陷分类的一种抽象策略。Anda 和 Sjøberg 在文献[115]中提出的用况缺陷分类可以看做是对 Denger[116]等人提出的缺陷分类针对用况模型的一种应用。Cox[6]等人以及 Phalp[117]等他们各自的缺陷分类侧重于用况规约中文本描述的语义错误。文献[116]和[115]都是通过分析不同类型的错误对用况质量的影响而进行的缺陷分类，而文献[6]和文献[117]是从错误源的角度定义了一系列用况规约评价指标。

本章阐述的 RUCM 用况缺陷分类要更加全面系统化，它是严格遵循工业标准 IEEE

Std. 830-1998[72]对 RUCM 用况模型进行的系统化定义，同时也借鉴了上述文献中的研究成果将它们具体应用到 RUCM 用况模型中。因此，RUCM 缺陷分类不仅涵盖了文献研究中的缺陷分类从而可以适用于常规的用况模型，而且针对 RUCM 模型元素给出了更加具体的缺陷类型定义。

4.5.3 使用 HAZOP 创建变异算子

HAZOP 是一种系统化的方法，已有研究者将这种方法引入到程序变异中进行变异算子的生成。 Kim 等人在他们的研究报告[132]中详细的阐述了他们针对 Java 编程语言是如何使用 HAZOP 系统化的创建 Java 变异算子的。在研究报告[132]中，作者针对 Java 程序规范说明设计了 10 个 HAZOP 关键字：*NO*、*MORE*、*LESS*、*AS WELL AS*、*PART OF*、*REVERSE*、*OTHER THAN*、*NARROWING*、*WIDENING*、*EQUIVALENT*。其中前七个关键字是标准[73]中定义的基本关键字，最后三个关键字是针对 Java 编程语言新设计的。关键字 *NARROWING* 和 *WIDENING* 用来处理 Java 程序中的访问控制权限，即面向对象中的封装和信息隐藏，例如 Java 关键字‘private’、‘public’的使用。关键字 *EQUIVALENT* 用于识别等价变种。

本章 RUCM 变异算子的创建是基于对 RUCM 模型元素的严格分析得出的。将 HAZARD[73]工业标准中的基本关键字(*NO*、*MORE*、*LESS*、*AS WELL AS*、*PART OF*、*REVERSE*、*OTHER THAN*)应用到每一个 RUCM 模型元素进行分析，然后依据相应的负影响创建对应的 RUCM 变异算子。

4.6 本章小结

软件需求的质量对于软件系统的成功开发是至关重要的。需求评审技术是已被实证研究证明的一种确保需求质量的有效技术，并在工业实践中被广泛使用。随着用况建模成为工业实践中广泛使用的一种捕获、描述系统需求的方法，针对用况模型的评审技术的研究成为研究学者的一个关注热点。然而，在目前的学术研究中存在两个明显的问题：1) 缺少一种系统化的创建用况模型缺陷的方法，从而无法全面的评价不同的需求评审技术的有效性；2) 需要一种针对用况模型的全面的缺陷分类，并且针对每一个缺陷类型要给出确切的定义。这样才可以准确的定义缺陷发现率从而对不同的评审技术进行有效的评估。

本章阐述了一种新颖的方法 MuRUCM 用以解决上述问题。通过对工业标准 IEEE Std. 830-1998 进行系统化的研究，定义了一个针对用况模型的全面的需求缺陷分类。通过严格遵循工业标准 HAZOP 对 RUCM 用况模型进行分析，创建了一系列的 RUCM 变异算子用以创建不同类型的需求缺陷。最后，给出了一组辅助规则用以创建缺陷生成策略。本章从不同的领域选用了不同复杂程度的 2 个工业案例和 9 个文献案例对提出的 MuRUCM 方法进行验证，实验结果显示：提出的需求缺陷分类适用于 RUCM 用况模型，生成的 RUCM 变种能够有效地区分三种不同的 RUCM 用况场景生成策略。

第5章 基于相似度函数和搜索算法的用况场景选择方法

本章的研究目标主要是解决如何在需求评审成本(可用时间、可投入人力)有限的情景下，从大量的用况场景中选择出其中的一部分作为评审源，期望能够包含尽可能多的软件需求问题，以支持需求评审的开展。为了解决这个工业实践问题，本章通过采用基于搜索的软件工程 (Search-based Softwre Engineering)方法将其转化为一个优化选择问题。为了对提出的选择方法进行充分可靠的有效评估，本章采用实证软件工程 (Empirical Softeware Engineering)的方法进行实证研究 (Empirical Study)。 本章内容组织如下：5.1 节对本章的研究问题进行详细的阐述；5.2 节详细论述了基于相似度函数和搜索算法的 RUCM 用况场景选方法 S^3RUCM；5.3 节将转化后的优化选择问题进行形式化定义并论述了适应度函数的设计；5.4 节中采用实证研究的方法通过工业案例和文献案例对提出的 S^3RUCM 进行全面的评估；5.5 节对相关的研究工作进行对比分析；5.6 节总结了本章的主要内容。

本章提出的用况场景选择方法 S^3RUCM 不仅适用于标准的 RUCM 用况模型，同时也适用于 RUCM4RT 用况模型(第 2 章)和 rtAspectRUCM 用况模型(第 3 章)，在本章的行文中不再对它们进行严谨区分，而把它们通称为 RUCM 用况模型。

5.1 研究问题的提出

软件需求在复杂实时系统的开发中扮演重要的角色[10]，而用况建模是一种被广泛使用的需求建模技术。通过将图形化的形象展示和精准的文字描述进行有效融合，用况模型为软件需求规约提供了一种直观而准确的建模方式。一个用况捕获了系统在特定条件下的一系列动作行为，它描述了系统如何与参与者进行交互并完成相应的系统功能。对于一个用况而言，其最重要的构成要素就是用况场景。每个用况场景描述了一定条件下系统执行的动作序列[10].

在复杂实时系统的工业实践中，一个用况可以诱发大量用况场景的产生，特别是当系统的用况模型中使用了大量的包含«Include»、扩展«ExtendIn» 关系。这种现象在安全关键软件 (Safety-Critical Sytem)的开发过程中尤为普遍，因为安全关键软件的需求开发不仅要对复杂的系统功能进行捕获描述同时更加关注对各种异常情形的处理，而这些异常情形的处理通常都会被建模为用况的备选事件流 (alternative flow-of-events)或异常事件流(exceptional flow-of-events)[133]。在用况驱动的软件开发过程中，用况场景通常为软件需求评审和分析提供输入，并为需求驱动的测试 (requirements-driven testing)和后续的

软件开发活动提供支持和依据[10,12]。对安全关键软件(如航空领域)而言，需求评审通常被作为保证最终产品质量的一个必要环节[134]。用况模型已被广泛应用于需求评审中[135,136,19,12]。手工需求评审要求领域专家对所有可用的需求描述进行走读，从而对其中存在的需求错误进行识别和修复[137]。在市场驱动开发 (Market-driven Development)的背景下，通常无法在有限的成本(时间和人力)内对所有的需求描述完成需求评审，常用的技术手段就是选择部分需求项进行评审[80]，而选择的标准或者是依据领域专家的经验或者是采取随机的策略。针对如何选择相应的需求(用况场景)以开展需求评审这个问题的研究目前还非常少。

为了解决上述挑战，本文提出一种自动化的基于相似度函数和搜索算法的方法以选择出一部分需求，从而为手工需求评审的开展提供支撑技术。在本章的语境下，需求即 RUCM 用况场景，因此，本章在论述 S^{3}RUCM 时“需求”和“用况场景”这两个术语可以交替使用。

5.2 S^{3}RUCM 用况场景选择方法

本文提出一种基于相似度函数和搜索算法的用况场景选择方法 S^{3}RUCM (similarity-based and search-based use case scenarios selection approach)。S^{3}RUCM 的主要目标是在用况场景的选择过程中对用况场景集的相似度进行最小化以期望发现尽可能多的需求缺陷。给定一个 RUCM 用况，由其产生的不同用况场景可能会包含一些相同的需求描述语句(RUCM step)。因此，在用况场景选择的过程中选择那些覆盖掉尽可能多的需求描述语句的用况场景将有助于发现更多的需求错误。为了形象的阐述 S^{3}RUCM 方法，本文采用业务流程建模标记法 BPMN[111] (Business Process Model and Notation (BPMN)) 并在基于 RUCM 的需求评审的环境下，对 RUCM 用况建模、用况场景选择和用况评审之间的关系进行阐述。

如图 31 所示，整个流程可以细化成三个过程。第一个过程是 RUCM[23]用况建模，通常由需求分析人员作为该过程的主要相关者，该过程主要是将软件需求建模为 RUCM 用况并通过 RUCM 需求模板进行详细描述。RUCM 包含了一个结构化的用况描述模板和一系列书写规则。RUCM 被设计来消除自然语言需求规约中难以避免的模糊性并支持从用况需求模型到 UML 分析模型(如 UML 类图)和测试用例的自动生成。RUCM 在描述高质量的软件需求[23]、实现由用况模自动转换成 UML 分析模型[8]、实现由用况模型自动生成测试用例[9]方面的有效性已被验证。RUCM 支持三种不同的用况场景生成策略，

语句覆盖(*All_Steps coverage criterion*)、事件流覆盖(*All_Flows coverage criterion*)、条件覆盖(*All_Conditions coverage criteria*)。如图 31 所示，该过程产生的制品是 RUCM 用况和相应的用况场景。该过程可以采用第二章和第三章的方法进行创建。

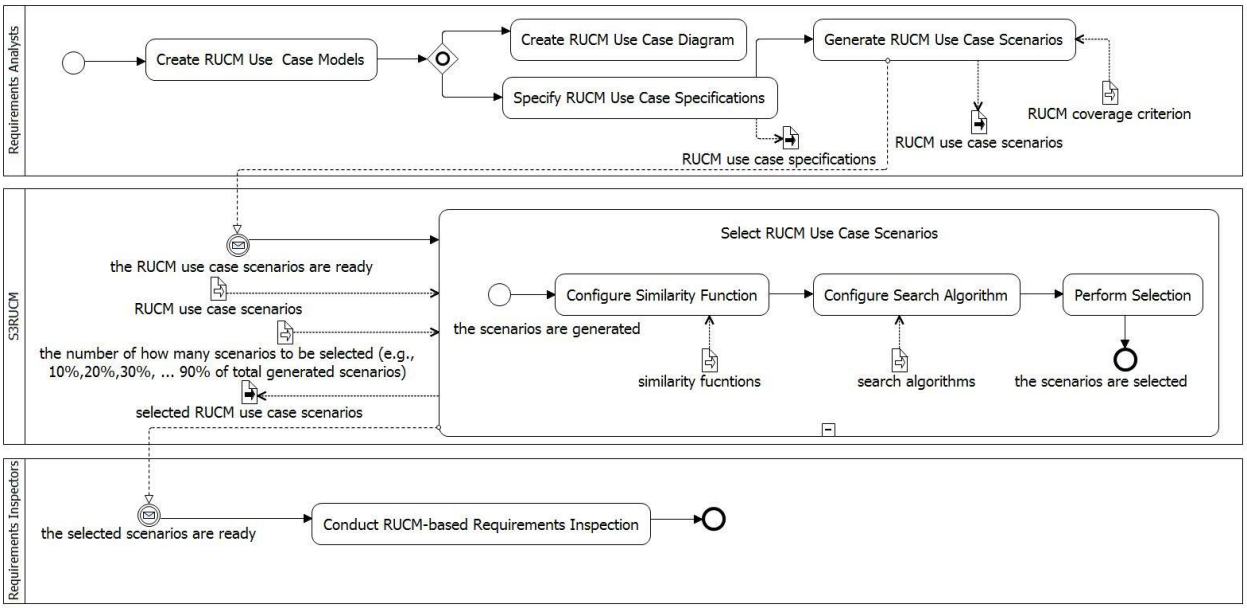

图 31 S³RUCM 概览图 (使用 BPMN 标记法)

在有限的评审成本(时间和人力)限制下，从自动生成的 RUCM 用况场景中选择出那些差异最大的用况场景就是第二个过程，该过程由 S³RUCM 自动实现。S³RUCM 采用相似度函数对任意两个用况场景之间的相似性进行计算，并使用搜索算法识别出平均相似度最小的那一组用况场景。为了解决这个优化选择问题，本文设计并实现了相应的适用度函数(fitness function)以指导搜索算法的有效进行。为了对设计的适用度函数进行有效评估，本文采用以下几种搜索算法：Genetic Algorithms (GAs)、 (1+1) Evolutionary Algorithm ((1+1)EA)、Alternating Variable Method (AVM)、Random Search (RS)。详细的算法信息在 5.2.3 节进行阐述。其中 *RS* 通常被用作“评估基线”用以验证所优化的问题不是一个朴素的问题，即使用搜索算法的必要性[138]。为了计算用况场景之间的相似性，本文采用以下八种相似度函数：Counting function (*CNT*)[139]、Jaccard Index (*JAC*)[140]、Gower-Legendre (*GOW*)[140]、Sokal-Sneath (*SOK*)[140]、Normalized Longest Common Subsequence (*NLCS*)、 Levenshtein Distance (*LEV*)[141]、Needleman-Wunsch (*NW*)[142]、Smith-Waterman (*SW*)[142]。以上各个相似度函数将在 5.2.2 节进行详阐述。搜索算法搭配相似度函数已经被采用来解决测试用例的优化选择问题[143-145]，然而，根据目前的文献调研所知还没有将搜索算法和相似度函数应用于用况场景的优化选择问题。

如图 31 所示，最后一个过程就是需求评审，通常由评审人员执行。在本文的研究中

采用实证研究(Empirical Study)的方法对 S^{3}RUCM 的有效性进行评估，而不是在于提出一种新的具体的评审技术。然而 S^{3}RUCM 实现的用况场景选择方法可以集成到不同的需求评审方法中(如 ad-hoc[137,146]、checklist[147,11,148])，当然这还需要进行更多的研究验证以确定其成本-效益(cost-effectiveness)。为了对 S^{3}RUCM 进行有效评估，实验中采用第五章阐述的 MuRUCM 方法系统化的创建各种需求缺陷。

5.2.1 自动生成 RUCM 用况场景

在本文中，一个 RUCM 用况场景就是一些顺序执行的 RUCM 描述语句(RUCM steps)，而这些 RUCM 描述语句只能构成 RUCM 用况的一个分支执行。一个 RUCM 用况场景只能从 RUCM 的基本流开始，它可能会跨越不同的 RUCM 备选流 (借助 RFS 关键字)，它要么返回基本流结束(借助 RESUME STEP 关键字)，要么在备选流中结束(由 ABORT 关键字指明)。图 32 和图 33 展示了一个使用 RUCM 方法进行描述的用况“Validate Pin”。

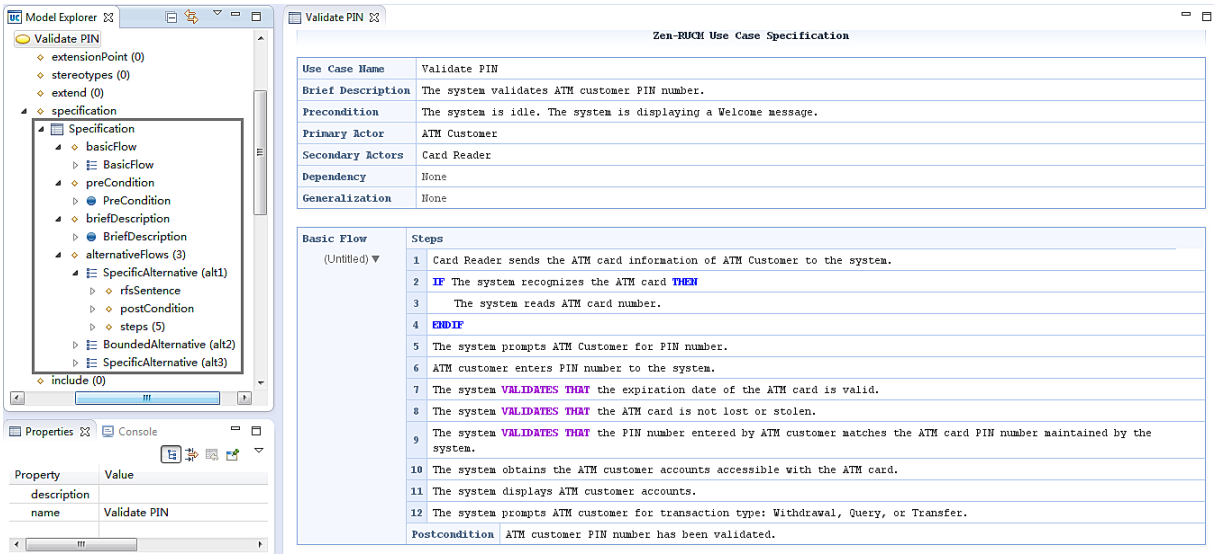

图 32 用况 *Validate Pin* 的 RUCM 需求描述 (RUCM 基本流)

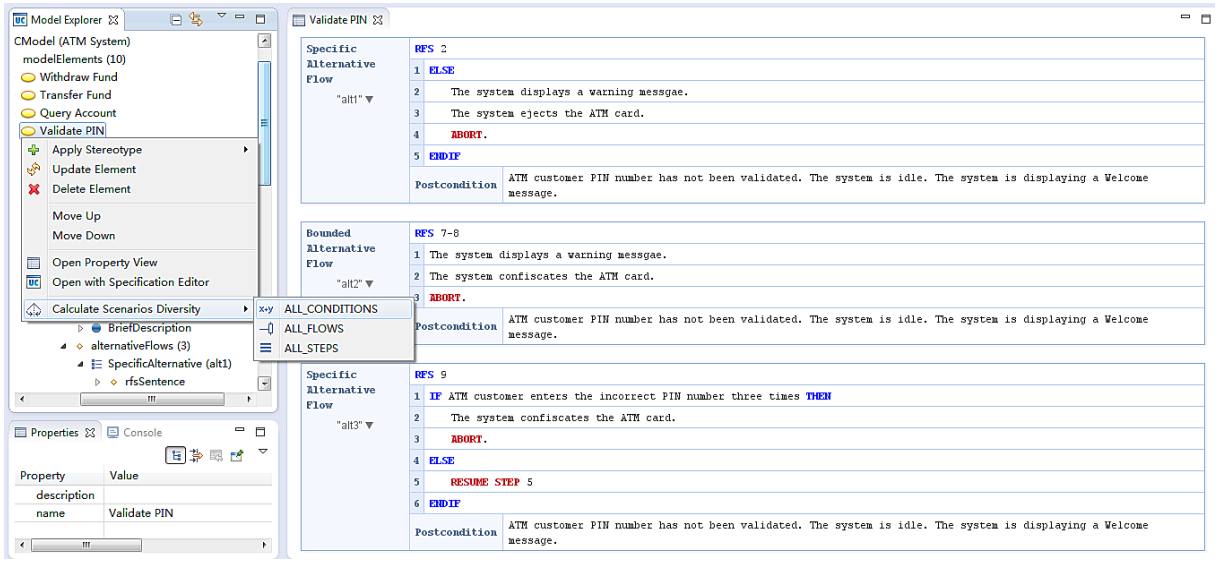

图 33 用况 *Validate Pin* 的 RUCM 需求描述 (RUCM 备选流)

每个 RUCM 用况都会被形式化为一个 UCMeta 实例，RUCM 借助自定义的 RUCM 事件流、关键字(如，DO-UNTIL, INCLUDE USE CASE)采用一种结构化的方法对需求描述中的控制流信息进行捕获并建模。基于 RUCM 用况模型中内嵌的控制流信息，RUCM 支持以下三种覆盖标准用以产生RUCM用况场景：全条件覆盖(*All Condition Coverage*)、全事件流覆盖(*All FlowOfEvents Coverage*)、全语句覆盖(*All Sentence Coverage*)。

全条件覆盖(*All Condition Coverage*)确保所有引起分支的 RUCM 条件描述语句都被覆盖并且被执行至少一次。对于用况描述中的迭代结构体(如 DO-UNTIL)，RUCM 在生成用况场景时会保证迭代结构体可以执行一次或 x 次，其中 x 是一个可配置的正整数。在本文的实验中，x 被设定为 1，即执行一次。 全事件流覆盖(*All FlowOfEvents Coverage*)确保 RUCM 基本流和所有的备选流至少被覆盖一次。对于一个特定流(specific flow)或者组界流(bounded flow)，借助于分支指示语句(RFS flow step)，往往会产生一个或多个 RUCM 分支。对于全局流(global flow)，RUCM 会针对 RUCM 基本流，选择一个描述语句创建相应的分支。全语句覆盖(*All Sentence Coverage*)确保所有的 RUCM 描述语句都被覆盖至少一次。以上覆盖策略已被用于生成测试用例[81,23]。

5.2.2 相似度计算函数

在本文的研究中，S^{3}RUCM 采用了 8 种不同的相似度计算函数，下面对它们进行详细阐述。为了形象准确的解释每个相似度函数的计算原理，本节首先从 RUCM 用况 *Validate Pin* (图 32 和图 33)中选取如下两个用况场景：

- Scenario 1: Basic flow step 1 (b_1)→Basic flow step 2 (b_2)→Basic flow step 3

(b_3)→Basic flow step 5 (b_5)→Basic flow step 6 (b_6)→Basic flow step 7 (b_7)→ alt2 step 1 ($alt2_1$)→ alt2 step 2 ($alt2_2$)→alt2 step 3 ($alt2_3$)

- Scenario 2: Basic flow step 1 (b_1)→Basic flow step 2 (b_2)→Basic flow step 3 (b_3)→Basic flow step 5 (b_5)→Basic flow step 6 (b_6)→Basic flow step 7 (b_7)→ Basic flow step 8 (b_8)→ alt2 step 1 ($alt2_1$)→ alt2 step 2 ($alt2_2$)→alt2 step 3 ($alt2_3$)

进一步，将以上两个用况场景进行如下的抽象编码：

- $S_1 = \{b_1, b_2, b_3, b_5, b_6, b_7, alt2_1,\ alt2_2,\ alt2_3\}$
- $S_2 = \{b_1, b_2, b_3, b_5, b_6, b_7,\ b_8, alt2_1,\ alt2_2,\ alt2_3\}$

5.2.2.1 基于集合的相似度函数

基于集合的相似度函数(Set-based similarity functions)广泛的应用于数据挖掘中，用来计算两个多维特征向量的数据对象之间的接近程度[141]。在本文中，每个 RUCM 用况场景可以建模为一个向量，向量中的元素就是一系列顺序执行的 RUCM 需求描述语句(RUCM steps)。 基于集合的相似度函数在计算相似度时并不考虑向量中各个元素出现的次序，而同一个元素的多次出现也仅被计算一次。由于 RUCM 用况场景包含的需求描述语句的个数不同，当 RUCM 用况场景被建模为向量后，向量的元素数也就不同。在接下来的阐述中，本文将 5.2.2 节中编码的 RUCM 用况场景 S_1 和 S_2 作为相似度计算的输入对各个基于集合的相似度函数进行阐述。S_1 和 S_2 有共同的元素集合 $\{b_1, b_2, b_3, b_5, b_6, b_7, alt2_1, alt2_2,\ alt2_3\}$，$S_1$ 的长度是 9 而 S_2 的长度是 10。

本文的研究中选取了 Counting function (CNT)[139]、Jaccard Index (JAC)[140]、Gower-Legendre (GOW)[140]、Sokal-Sneath (SOK)[140] 四种不同的相似度计算函数。如文献[140]中所论述的，JAC、GOW、SOK 是最常用的用于比较两个对象的相似度的方法。CNT 是在研究工作[139]中被提出的，它被设计来进行测试用例的选择。本文中将 CNT 设计成一个通用的版本。所有这些基于集合的相似度函数，它们从不考虑元素的出现次序和重复次数，这是它们与基于序列的相似度函数之间的最大不同。本文中采用的基于集合的相似度函数已被用来解决测试用例的选择优化问题[145,149]。

- Counting Function (CNT)

CNT 是从文献[139]中借鉴来的，在文献[139]中 CNT 被设计来针对标记变迁系统(Labeled Transition Systems)中的迁移进行成对比较。本文将原有的 CNT 定义进行了重构：两个输入向量中相同的元素数与向量的平均长度的比值。给定两个 RUCM 用况场景 A 和 B，CNT 的计算公式如下：

$$CNT(A,B)=\frac{|A\cap B|}{(|A|+|B|)/2} \quad (1)$$

其中 $|A\cap B|$ 是 A 和 B 交集的元素个数，即两个 RUCM 用况场景中包含的相同的 RUCM steps；$|A|$用于计算 A 中的元素个数，即 RUCM 用况场景中包含的不同的 RUCM steps 的总数。假定以S_1和S_2作为输入，$CNT(S_1,S_2)\ =\ 9\ /\ ((9+10)\ /\ 2)\ =\ 0.9474$。

- Jaccard Index (JAC) [140]

JAC 的计算公式如下：

$$JAC(A,B)=\frac{|A\cap B|}{|A\cup B|}=\frac{|A\cap B|}{|A\cap B|+(|A\cup B|-|A\cap B|)} \quad (2)$$

- Gower-Legendre (GOW) [140]

GOW 可以看作是 JAC 的一种变体，其计算公式如下：

$$GOW(A,B)=\frac{|A\cap B|}{|A\cap B|+\frac{1}{2}*(|A\cup B|-|A\cap B|)} \quad (3)$$

- Sokal-Sneath (SOK) [140]

SOK 是 JAC 的另一种变体，其计算公式如下：

$$SOK(A,B)=\frac{|A\cap B|}{|A\cap B|+2*(|A\cup B|-|A\cap B|)} \quad (4)$$

JAC、GOW 和 SOK 之间的区别在于它们各自对于两个输入集合中不同元素(即，$|A\cup B|\ -\ |A\cap B|$)的权值的不同。其中 JAC 是 1，GOW 是$1/2$，SOK 是 2。对于同样的输入，GOW 的相似度数值大于 JAC 的相似度数值，而 JAC 的相似度数值大于 SOK 的相似度数值。假定以S_1和S_2为输入，$|\ S_1\cup S_2\ |=10$，$|\ S_1\cap S_2\ |\ =\ 9$，$JAC(S_1,\ S_2)=9/10=0.9$，$GOW(S_1,\ S_2)=18/19=0.9474$，$SOK(S_1,\ S_2)=9/11=0.8182$。

5.2.2.2 基于序列的相似度函数

对基于序列的相似度函数(sequence-based similarity functions)，两个输入序列被当作编辑距离进行处理[141]。编辑距离被定义为将一个序列转变成另一个序列所需要的最少的操作，其中编辑操作包含插入、删除、替换[141]。与基于集合的相似度函数不同，输入序列中元素的次序和重复次数都会影响相似度数值的计算。汉明距离 Hamming Distance (HD)[150]是一种著名的基于序列的相似度计算函数，然而，它限定输入的序列的必须是等长的。因此，它在本文的 RUCM 用况场景选择问题中并不适用，因为不同的 RUCM 用况场景可能包含不同数目的 RUCM 描述语句。基于编辑距离，可以通过对两个输入序列中元素匹配进行奖励，同时惩罚元素不匹配和空缺，设计相应的相似度函数对两个

输入序列进行相似度计算。本文采用了四种不同的基于序列的相似度函数：Normalized Longest Common Subsequence (NLCS)、Levenshtein Distance (LEV)[141]、Needleman-Wunsch (NW)[142]、Smith-Waterman (SW)[142]。LEV、NW、SW 在生物信息学中被广泛使用，它们现在也被应用到测试用例的选择中[145]。最长公共子序列 Longest Common Subsequence (LCS) 是一种用于确定两个序列是否相关的常用技术[141]。基于最长公共子序列 LCS，本文定义了标准化的最长公共子序列(Normalized Longest Common Subsequence (NLCS)) 用以计算两个序列的相似度数值。区别于基于集合的相似度函数，符号 '| |' 在本小节的公式中的应用指明了一个 RUCM 用况场景中所包含的所有需求描述语句包含同一个语句的多次出现。

- Normalized Longest Common Subsequence (NLCS)

最长公共子序列 LCS 是一类经典的计算机科学问题，它被广泛应用于生物信息学中。LCS 并不要求子序列必须是在原序列中占据连续位置的。为了将 LCS 可以适用于计算相似度数值，本文对 LCS 进行了扩展，即标准化的 LCS (Normalized Longest Common Subsequence (NLCS))。NLCS 被定义为：LCS 的长度与输入序列平均长度的比值。NLCS 的计算公式如下：

$$NLCS(A,B) = \frac{LCS'_{A,B}}{(|A| + |B|)/2} \quad (5)$$

其中$|A|$或$|B|$用于计算 RUCM 用况场景的长度，即包含的所有 RUCM 需求描述语句。$LCS'_{A,B}$ 代表 LCS 的长度，其计算公式如下：

$$\underset{0\le i\le|A|,0\le j\le|B|}{LCS_{A,B}}(i,j) = \begin{cases} 0, & if\ \ i = 0\ or\ j = 0 \\ LCS_{A,B}(i-1,j-1) + 1, & if\ A_i = B_j \\ max(LCS_{A,B}(i-1,j), LCS_{A,B}(i,j-1)), & if\ A_i \neq B_j \end{cases} \quad (6)$$

其中A_i是用况场景 A 中第 i 个需求描述语句，B_j 是用况场景 B 中第 j 个需求描述语句，i 和 j 非别是 A 和 B 的最大长度。 假定以S_1和S_2为输入的序列，则$LCS'(S_1,S_2) = 9$ 其最长公共子序列 LCS："$b_1, b_2, b_3, b_5, b_6, b_7, alt2_1,\ alt2_2,\ alt2_3$"，因此，$NLCS(S_1,S_2) = 9/((9+10)/2) = 0.9474$。

- Levenshtein Distance (LEV) [141]

LEV 距离被定义为在序列转换过程中每做一次替换(即元素不匹配)或者进行一次删除/插入操作都会将距离增加一个计量单位[141]。在本文中，给定两个 RUCM 用况场景 A 和 B，$LEV_{A,B(|A|,|B|)}$ 的计算公式如下：

$$\underset{0\le i\le|A|,0\le j\le|B|}{LEV_{A,B}}(i,j)=\begin{cases}max(i,j), & if\ min(i,j)=0\\ min\begin{cases}1+LEV_{A,B}(i-1,j)\\ 1+LEV_{A,B}(i,j-1)\\ 1_{A_i\neq B_j}+LEV_{A,B}(i-1,j-1)\end{cases}\end{cases} \quad (7)$$

假定以S_1和S_2作为输入，S_2中包含一个S_1中所没有的元素‘b_8’；因此，相应的 LEV 距离就是 1，即将S_1转变成S_2或者将S_2转变成S_1至少要进行一次编辑操作。

➢ 全局序列比对方法 Needleman-Wunsch (NW)[142]

序列比对用于对两个输入序列中包含的元素进行匹配映射[142]，序列比对算法的目标是发现一种对输入序列的元素进行安置定位的方式从而使得比对数值可以达到最大化。比对数值是用来表征元素匹配、元素不匹配、元素空缺的，它实际上就是两个序列的相似度数值。全局序列比对算法(Global alignment)是对输入序列进行整体比对，其中被最广泛使用的算法是 Needleman-Wunsch (NW)[142]。本文中对 NW 过程中的操作权限设置如下： 元素匹配 +1，元素不匹配 -1、元素空缺 -1。 NW 的权重矩阵 F 定义如下：

$$F_{[r][0]}=r*d\ where\ 1\le r\le i;\ F_{[0][c]}=c*d\ where\ 1\le c\le j\ ;i=|A|,j=|B|;$$

$$F_{[r][c]}=max\begin{cases}F_{[r-1][c-1]}+sim(A_r,B_c)\\ F_{[r-1][c]}+d\\ F_{[r][c-1]}+d\end{cases} where\ 1\le r\le i,1\le c\le j;i=|A|,j=|B| \quad (8)$$

其中 A 和 B 是输入序列，函数$sim(A_r,\ B_c)$针对 A 中的第r个的元素和 B 中的第c个元素返回这两个元素的匹配数值或者不匹配数值，d 是元素空缺的惩罚数值。A 和 B 之间的相似度数值就是$F_{[i][j]}$，即矩阵 F 中的最后一个元素，i 是 A 的长度，j 是 B 的长度。因此，NW 的相似度函数定义如下：

$$NW_{A,B}=F_{[i][j]}\ where\ i=|A|,j=|B| \quad (9)$$

类似的，当以 S_1 和 S_2 作为输入序列，$NW(S_1,S_2)=8$。 具体讲， 经过 NW 全局比对后得到的序列操作如下：

$$\begin{pmatrix}b_1,b_2,b_3,b_5,b_6,b_7,\ -,\ alt2_1,\ alt2_2,\ alt2_3\\ b_1,b_2,b_3,b_5,b_6,b_7,\ \ b_8,alt2_1,\ alt2_2,\ alt2_3\end{pmatrix},$$

其中包含九个匹配(即，$b_1,b_2,b_3,b_5,b_6,b_7,alt2_1,\ alt2_2,\ alt2_3$)，一个空缺(即，‘$-$’, b_8) ，零个不匹配，因此$NW(S_1,S_2)=(+1)*9+(-1)*1=8$。

➢ 局部序列比对方法 Smith-Waterman (SW) [142]

局部序列比对的目标是从输入序列的所有子序列对中发现能够取得最大比对值的那一组子序列。局部序列比对算法的返回结果是找到的子序列对和相应的最大比对数值。Smith-Waterman (SW) [142]算法是应用最广泛的局部比对算法[142]，SW 的权重矩阵 F 定义

如下：

$$F_{[r][0]} = 0\ where\ 1 \le r \le i;\ F_{[0][c]} = 0\ where\ 1 \le c \le j; i = |A|, j = |B|;$$

$$F_{[r][c]} = max \begin{cases} F_{[r-1][c-1]} + sim(A_r, B_c) \\ F_{[r-1][c]} + d \\ F_{[r][c-1]} + d \end{cases} \quad where\ 1 \le r \le i, 1 \le c \le j; i = |A|, j = |B| \quad (10)$$

在 SW 的权重矩阵 F 中，每个矩阵元素的数值都是非负的，最终的相似度数值是矩阵$F_{[r][c]}$中数值最大的那一个。因此，SW 的相似度函数计算公式定义如下：

$$SW_{A,B} = \max_{1 \le r \le i, 1 \le c \le j} (F_{[r][c]})\ where\ i = |A|, j = |B| \quad (11)$$

其中 i 是 A 的长度，j 是 B 的长度，F 是公式 (10) 中定义的矩阵。

假定以S_1和S_2作为输入的序列，SW 比对后得到的序列操作如下：

$$\begin{pmatrix} b_1, b_2, b_3, b_5, b_6, b_7, -, alt2_1, alt2_2, alt2_3 \\ b_1, b_2, b_3, b_5, b_6, b_7, b_8, alt2_1, alt2_2, alt2_3 \end{pmatrix},$$

其中包含九个匹配(即，$b_1, b_2, b_3, b_5, b_6, b_7, alt2_1, alt2_2, alt2_3$)，一个空缺(即，‘$-$’, b_8) ，没有不匹配，因此$SW(S_1, S_2) = (+1) * 9 + (-1) * 1 = 8$。

5.2.3 启发式搜索算法

在本文的研究中，如图 31 所示，给定一个具体的相似度函数和期望的用于需求评审的 RUCM 用况场景的数量 m (通常是所有用况场景数量的一个百分比，并且由领域专家预先给定)，搜索算法将会从所有的用况场景中选择出数量是 m 的一组用况场景，并且保证这一组用况场景的平均相似度数值是所有可能构成 m 个用况场景的候选组合中最小的那一组。在基于搜索的软件工程[151]的实践中，遗传算法 Genetic Algorithms (GAs)，爬山算法 Hill Climbing 和进化算法 Evolutionary Algorithms (EAs)是常用的用于解决优化问题的技术，S^{3}RUCM 使用这些算法来解决用况场景选择的问题。给定一个具体的相似度函数，S^{3}RUCM 支持四种不同的搜索算法进行用况场景的选择，表 23 以伪码的形式展示了本文所采用的四种搜索算法，它们的具体阐释如下。

- 交替变量方法 Alternating Variable Method (AVM)

AVM 在文献[152]中被提出并已被证明是一种用于解决不同的软件工程问题的局部优化技术[153]。AVM 的输入数据被组织成一个向量，AVM 从搜索空间中随机的挑选一个起始点来启动搜索过程(表 23)。在每一轮的搜索过程中，AVM 都会针对输入向量中的每一个元素的所有取值进行探索性搜索。在对一个元素进行探索性搜索的过程中，其他的元素的取值保持不变。如果一个局部最优值找到了，AVM 将会进行下一个元素的搜

索。如果对所有的元素都完成一轮搜索后，适应度函数返回的数值并没有得到改进，AVM 将会重新选择一个随机起始点进行另一轮的搜索。 最后，当获得了最优的适用度函数值或到达搜索停止标准(例如，最大的搜索次数或者最长搜索时间)，AVM 便结束整个搜索过程。

- 稳态遗传算法 Steady-State Genetic Algorithms (SSGA)

遗传算法 Genetic Algorithms (GAs)[154]是受进化论启发而产生的。自从文献[155]设计出第一个适用模型，遗传算法被广泛的用作一种有效的优化技术。遗传算法依赖于三个算子：选择、交叉和变异。基于不同的创建种群数量的策略，遗传算法可以分为两大类[156]：Generational Genetic Algorithm (GGA)，该算法中每次产生新的子代时整个种群都会被同时替换掉； Steady State Genetic Algorithm (SSGA)，该算法中种群会重叠，即，子代创建过程中只会选择一个或两个新生成的孩子替代原来父母中最差的那个[157]。本文研究中采用了 SSGA 而不是 GGA 进行实证研究。因为相比较 GGA 而言，SSGA 需要更少的计算(即每一次迭代的搜索成本)同时也有更好的表现[158]。本文的 SSGA 算法中采用 simulated binary crossover (SBX)[159] 交叉算子和多项式变异 polynomial mutation operation[159]来创建新的孩子成员。只有那些比父母优异的孩子会被保留下来创建下一个子代(表 23)。在选择父母时，采用排序选择算子 rank selection[160]同时选用一种标准的替代策略(即删除种群中最差的那个成员)。表 23 描述了 SSGA 的搜索过程。

- (1+1)进化算法 (1+1) Evolutionary Algorithm ((1+1) EA)

(1+1) EA[161] 是一种单个个体的进化算法。(1+1) EA 是进化算法中最简单的一种变体，它的种群中只含有一个个体，而且也不采用交叉算子。(1+1) EA 从一个个体(即搜索空间中的一个起始点，通常是随机选取的)开始其搜索过程，然后在每一代的创建过程中通过对父亲进行变异操作产生唯一的一个孩子，最后从孩子和父亲之间选择一个最好的作为子代的唯一个体。表 23 描述了(1+1) EA 的搜索过程。

- 随机搜索 Random Search (RS)

RS 是最简单的搜索算法。它随机的从搜索空间中挑选候选方案并返回整个搜索过程中取得最好适应度函数值的那个方案。RS 在搜索过程中并不会利用已经搜索过元素的信息来指导下一次的搜索。RS 通常被用作基线用以比较其他启发式搜索算法的性能表现[162,163]。

表 23 四种不同搜索算法的伪码展示

AVM	***SSGA***

Repeat until the stopping criterion is met
Randomly select a set of use case scenarios: A(A_1, A_2, …, A_m), m=|A|;
Let itr:= 0, i:= 1
While (itr < m) do
let A′:= exploratory_search (A, A_i)
if fitNess(A′) < fitNess(A)
let A := A′; itr := 0
else
let itr:= itr + 1
let i:= (i+1) mod m
	Sample a population *G* of use case scenarios
	Repeat until the stopping criterion is met
	Choose *Offx* and *Offy* from *G*
	(Off_x',Off_y') = Crossover(Off_x, Off_y, Pc)
	Mutate(Off_x',Off_y',Pm)
	Off = Min(Off_x',Off_y')
	Wr = Worst(G)
	if (fitNess(Off) < fitNess(Wr))
	Wr = Off

(1+1) EA let P_m := 1/n; mutation probability Generate randomly an individual E_x Repeat until the stopping criterion is met E_x'=Mutate(E_X, Pm) if (fitNess(E_x') <= fitness(E_X)) $E_X = E_x'$	***RS*** candidates= { }, list of use case scenarios best = Double.MAX, the smallest fitness value Repeat until the stopping criterion is met Select randomly a subset of use case scenarios: *randSet* if (best < fitNess(*randSet*)) best = fitNess(*randSet*) candidates = randSet

在本文的研究，正如同文献[163]所提倡的，RS 被选用作基线用以比较其他的搜索算法。AVM 作为局部优化技术的代表而被选用，而 SSGA 和 (1+1) EA 则代表着全局优化的技术。对所有的搜索算法，采用统一的停止标准，即 25000 次的优化比较。

5.3 搜索优化问题的形式定义和适应度函数

在基于搜索的软件工程实践中，为了能够有效的使用搜索技术必须要针对具体的工程问题进行分析，将其转化成一个搜索优化问题，然后再针对该搜索优化问题设计相应的适应度函数。本节将从这两方面对 5.2 节提出的 S^3RUCM 方法进行相应的阐述。5.3.1 节对 S^3RUCM 所要解决的搜索优化问题进行详细阐述和形式定义；5.3.2 节阐述了相应的适应度函数的设计。

5.3.1 搜索问题的形式定义

该小节论述了如何将 RUCM 用况场景选择问题形式化地定义为一个搜索问题。

5.3.1.1 基本概念定义

问题背景描述。如 5.2.1 节所论述，RUCM 用况建模方法提供了一个结构化的用况模板。RUCM 采用 UCMeta 元模型的形式化方式，将一个 RUCM 模型形式化成一系列 UCMeta 的元模型实例，从而实现三种不同覆盖方式的 RUCM 用况场景的自动生成。具体地讲，给定一个 RUCM 用况规约ucs， ucs可以产生一个相应的 RUCM 用况场景集

合，其中的每一个 RUCM 用况场景都封装了一系列顺序执行的 RUCM 需求描述语句。不同的 RUCM 用况场景可能包含一些相同的 RUCM 需求描述语句。假定正整数m代表了将要用于需求评审的 RUCM 用况场景的数量，S^{3}RUCM 的目标是从所有的 RUCM 用况场景集合中挑选出m个，并且保证这m个用况场景的平均的相似度数值是最小的。在 S^{3}RUCM 中，目标 RUCM 用况场景的数量m是可配置的。具体地讲，给定一个ucs和相应的用况场景覆盖标准，其生成的 RUCM 用况场景的总数是确定的，假定是n。如果目标 RUCM 用况场景的数量是m，即需要从n个 RUCM 用况场景中选择出m个进行需求评审且$m \leq n$，那么 S^{3}RUCM 将会从n个 RUCM 用况场景中选择出m个平均相似度数值最小的 RUCM 用况场景。在本文的实证研究中，将m实例化为实验设计中的一个自由变量'NUCS'(5.4.3.3 节)，它表示所产生的 RUCM 用况场景总数量的百分比，即 10%、20%、30%、40%、50%、60%、70%、80%、90%。 为形式化定义搜索问题，给出下面几个基本概念。

定义 5.1. $SN_{ucs} = \{s_1,\ s_2, \ldots, s_n\}$ 代表一个 RUCM 用况场景集合。给定一个 RUCM 用况规约ucs，SN_{ucs}将会依据不同 RUCM 覆盖标准(即，全条件覆盖(*All Condition Coverage*)、全事件流覆盖(*All FlowOfEvents Coverage*)、全语句覆盖(*All Sentence Coverage*))自动生成，其中n是生成的 RUCM 用况场景的总数量，s_i是一个具体的 RUCM 用况场景。一个 RUCM 用况场景s_i就是一系列顺序执行的 RUCM 需求描述语句。由于不同的用况场景可能包含相同的 RUCM 需求描述语句，即$|s_i \cap s_j| \geq 0$, 因此，对于一组 RUCM 用况场景就存在一定的相似度水平，在 S^{3}RUCM 中采用相似度函数(如，LEV、SW、NW)来进行计算。

定义 5.2. $SimFun(s_i,\ s_j)$ 表示一个相似度函数，它用来计算任意两个不同的 RUCM 用况场景s_i和s_j的相似度数值。在本文的研究中，S^{3}RUCM 采用了八种不同的相似度函数，即，$SimFun \in \{CNT, JAC, GOW, SOK, NLCS, LEV, NW, SW\}$。

定义 5.3.1. 给定一个 RUCM 用况场景集合SN_{ucs}，每一个选择方案可以形式化定义为$sol_i = \{s_1,\ s_2, \ldots, s_m\}$，即$sol_i$是由$m$个来自$SN_{ucs}$中的 RUCM 用况场景所构成。

定义 5.3.2. $SOL_m = \{sol_1,\ sol_2,\ sol_3,\ \ldots,\ sol_{|SOL_m|}\}$表示所有的选择方案。具体讲，给定一个 RUCM 用况规约ucs，依据不同 RUCM 覆盖标准其产生的 RUCM 用况场景集合SN_{ucs}是确定的，SOL_m就是所有那些可以从SN_{ucs}中选择出m个 RUCM 用况场景的选择方案的组合集。其中$|SOL_m|$表示所有选择方案的数量，即$|SOL_m| = \binom{|SN_{ucs}|}{m}$。由于搜

索空间非常庞大采用穷尽的探索方式不是一种切实可行的方法。例如，在本文的实证研究中，其中的一个案例的搜索空间是从 310 个 RUCM 用况场景中挑选出 31 个，即 $|SOL_m| = 310$，$m = 31$，$\binom{310}{31} \approx 4.3981 * 10^{42}$，这是一个非常大的搜索空间，采用穷尽的搜索方式对$SOL_m$中的每个方案进行验证是不可行的。

定义 5.4. $Similarity(sol_i)$ 用于计算选择方案sol_i的平均相似度数值。首先对sol_i中每一对 RUCM 用况场景进行相似度计算，然后再对所有的用况场景对的相似度数值计算平均值。相应的计算公式在 5.3.1.2 进行详细论证。

5.3.1.2 搜索问题的形式化定义

基于以上的定义 5.1-5.4，对本文中的 RUCM 用况场景选择问题进行如下的形式化定义。

搜索优化问题：给定一组 RUCM 用况场景SN_{ucs}和目标数量m，(即需要m个用况场景参与需求评审)，从SOL_m中选择一个候选方案sol_k以达到平均相似度数值的最小化：

$$\forall\, sol_j \in SOL_m : Similartiy(sol_k) \le Similartiy(sol_j)$$

5.3.2 适应度函数

为了有效的指导搜索算法选择出优化方案，需要根据优化选择问题设计相应的适应度函数。本节详细的阐述如何为 S^{3}RUCM 设计有效的适应度函数。如定义 5.4 中所述，$Similarity(sol_i)$计算选择方案sol_i的平均相似度数值。其详细的计算公式如下：

$$Similarity(sol_i) = \frac{\sum_{j=1}^{m-1}\sum_{k=j+1}^{m} SimFun(s_j, s_k)}{C_2^m},$$

$$SimFun \in \{CNT, JAC, GOW, SOK, NLCS, LEV, NW, SW\} \quad (12)$$

其中m是目标数量，即$m = |sol_i|$，表示需要选择m个 RUCM 用况场景参与需求评审；$SimFun$代表了所有的相似度计算函数；s_j和s_k表示选择方案sol_i中任意两个不同的 RUCM 用况场景。

不同的相似度计算函数有各自不同的取值范围，其中 CNT、JAC、GOW、SOK、NLCS 的取值范围是[0.0, 1.0]，NW 可以返回任何的整数，而 LEV 和 SW 可以返回任何的正整数。本文设计了公式(13)和公式(14)，用以把不同相似度函数的返回值进行标准化并统一规约到[0.0, 1.0]。经过这种标准化后，$stdSimilarity(sol_i)$的返回数值越大则说明方案sol_i的平均相似度越高。适用度函数的具体的定义过程如下。

针对 CNT、JAC、GOW、SOK 和 NLCS 这些相似度函数，直接采用它们计算的平

均相似度数值，因为这些数值的取值范围都是[0.0, 1.0]。针对 SW 相似度函数，先使用公式 $Nor(x)=\frac{x}{x+1}$[164]对返回的平均相似度数值进行标准化。

针对 LEV，使用$1.0-Nor(Similarity(sol_i))$对 LEV 的返回值进行标准化，因为 LEV 的返回值表示将一个 RUCM 用况场景转换成另外一个 RUCM 用况场景所使用的最小操作数，因此 LEV 的返回值越大则两个用况场景的相似度越小。

针对 NW 相似函数，采用公式(14)将 NW 的返回值规约到[0.0, 1.0]。 NW 的原始返回值可以是任何整数，包含负数、零、正整数，并且 NW 的返回值越小则说明两个 RUCM 用况场景越不相似。为了使得这些返回数值可以跟其他的相似度函数返回值进行比较，即适应度函数的取值范围是[0.0, 1.0]并且数值越小越好，公式(14)被设计来对 NW 的原始返回数据进行调整，并使用了标准化函数$Nor(x)=\frac{x}{x+1}$[164]。具体地讲，$0.5-0.5*Nor(-Similairty(sol_i))$将 NW 返回的负整数值规约到(0.0, 0.5)；NW 返回的正整数数值经过$0.5+0.5*Nor(Similairty(sol_i))$ 被规约到(0.5, 1.0)；最后，如果 NW 的原始返回值是 0.0，则将相应的适应度函数值直接赋值成 0.5。借助公式(13)和公式(14)，可以确保所有的相似度函数返回的数值的取值范围是[0.0, 1.0]并且数值越大说明相似度越高。

$$stdSimilarity(sol_i)=\begin{cases}Similairty(sol_i), & (CNT,JAC,GOW,SOK,NLCS)\\ Nor(Similarity(sol_i)), & (SW)\\ 1.0-Nor(Similarity(sol_i)), & (LEV)\end{cases} \quad (13)$$

$$stdSimilarity(sol_i)=\begin{cases}0.5+0.5*Nor(Similairty(sol_i)), & Similairty(sol_i)>0\\ 0.5, & Similairty(sol_i)=0\\ 0.5-0.5*Nor(-Similairty(sol_i)), & Similairty(sol_i)<0\end{cases} \quad (NW) \quad (14)$$

最后，适应度函数被定义为公式(15)。本文中，适用度函数主要用于指导搜索算法发现一组 RUCM 用况场景其平均相似度数值是最小的，因此在搜索过程中适用度函数的返回值越小说明当前的选择方案越好。

$$Fitness(sol_i)=\begin{cases}0.5+0.5*(\frac{DN}{m}+stdSimilarity(sol_i)), & (1\le DN<m;2\le m\le|SN_{ucs}|)\\ 0.5*stdSimilarity(sol_i), & (DN=0)\end{cases} \quad (15)$$

由于 S^{3}RUCM 需要搜索出m个不同的 RUCM 用况场景，公式(15)中设计了一种惩罚策略用于避免同一个sol_i中存在重复的RUCM用况场景。给定一个RUCN用况规约SN_{ucs}，

m是目标数量且$2 \le m \le |SN_{ucs}|$，并且m是可配置的参数，因此m决定了候选方案sol_i中不同的 RUCM 用况场景s_i的数量，即$|sol_i| = m$。DN用于表示当前方案sol_i中所有的重复的 RUCM 用况场景的总量，$DN = m - Different(sol_i)$其中$Different(sol_i)$ 是sol_i中所有不同的 RUCM 用况场景s_i的总量。例如，一个sol_i中含有 23 个 RUCM 用况场景，其中只有 15 个 RUCM 用况场景是彼此不同的，那么$DN = 23 - 15 = 8$。通过引入DN，将$\frac{DN}{m}$设计为一个惩罚因子添加到标准化后的平均相似度数值中，这样sol_i中包含的重复的 RUCM 用况场景越多将会导致最后的适应度函数值越大。此外，针对那些包含重复的 RUCM 用况场景的选择方案sol_i添加另一种惩罚措施，即将 0.5 添加到$0.5 * (\frac{DN}{m} + stdSimilarity(sol_i))$中。如公式(15)所示，如果$sol_i$包含重复的 RUCM 用况场景，则相应的适用度函数值经过$0.5 + 0.5 * (\frac{DN}{m} + stdSimilarity(sol_i))$计算处理，取值范围是(0.5, 1.0]；如果$sol_i$没有重复的 RUCM 用况场景，则相应的适用度函数值经过$0.5 * stdSimilarity(sol_i)$计算处理，取值范围是[0.0, 0.5]。通过这样做可以加速搜索过程以选择出m个不同的 RUCM 用况场景并且其平均相似度数值最小。本文中采用的惩罚策略是从文献[165]中借鉴的。

5.4 实证研究(Empirical Study)

为了对本文提出的 S^{3}RUCM 方法进行全面可靠的评估，本节进行实证软件工程(Empirical Software Engineering)中的实证研究(Empirical Study)。本节主要遵循经典的实证软件工程文献[74]提出的实证研究过程和[75,76]给出的指导规则开展相应的实证研究。5.4.1 节描述了本章的实验研究目标，5.4.2 节详细阐述了本章设计的实验研究问题，5.4.3 节对实验设计细节进行了详细报告，5.4.4 节说明了实验的具体实施，5.4.5 节汇报实验结果并进行相关分析，5.4.6 节对实验进行相关讨论，5.4.7 节对有效性风险进行讨论。

5.4.1 研究目标

本文的研究目标之一是择优选择部分用况场景以支持基于 RUCM 的用况评审。为了实现这个终极目标，本章提出了 S^{3}RUCM 方法，用以从大量的 RUCM 用况场景中选择出一部分并期望它们能包含尽可能多的需求缺陷。本节采用 *G-Q-M*(Goal-Question-Metric)[128]方法对实验目标进行如下的形式化阐述。

Goal 1。*从***支撑需求评审****的视角出发，以***评估****为目标，针对***其相应的效能和效率****，在***学术实验室****的环境下，对* **S^{3}RUCM** *进行分析。*

Goal 2。*从***支撑需求评审***的视角出发，以***评估***为目标，针对***彼此之间是否存在相关性***，在***学术实验室***的环境下，对***搜索算法选择出的 RUCM 用况场景集合的平均相似度和这组用况场景的缺陷发现率***进行分析*。

Goal 3。*从***支撑需求评审***的视角出发，以***评估***为目标，针对***时间效率***，在***学术实验室***的环境下，对***每个搜索算法的平均执行时间***进行分析*。

S^3RUCM 通过采用相似度函数和搜索算法来选择一组 RUCM 用况场景，并期望这组用况场景可以覆盖掉尽可能多的需求缺陷。其中 8 种相似度计算函数和 4 种搜索算法被采用，因此在接下来的实验中需要对这 32 种组合进行评估。为了设计并进行全面的有效评估，本文将上述 3 个实验研究目标 Goal 1-Goal 3 细化成一系列的实验研究问题，并在下节进行详细阐述。

5.4.2 研究问题

为了对 S^3RUCM 进行全面有效的评估，本节基于实验研究目标 Goal 1-Goal 3 设计如下的实验研究问题。

RQ1：针对每一种相似度函数，任意一种搜索算法与 RS 相比较，它是否在选择一组不相似的 RUCM 用况场景和需求缺陷发现率(DDR)两方面更加有效？

需求缺陷发现率(DDR)的具体计算公式在 5.4.3.3 小节进行阐述。这个研究问题的必要性在于：当优化问题比较简单时，随机搜索算法 RW 也会表现出比较好的性能，因此，需要首先将选择的搜索算法与随机搜索 RS 进行对比，从而说明搜索算法所要解决的问题不是一个朴素问题。

RQ2：针对每一种相似度函数，S^3RUCM 中采用的搜索算法在选择一组不相似的 RUCM 用况场景和需求缺陷发现率(DDR)两方面的效能如何？

这个研究问题的设计目标是要从所有的搜索算法中选择出效能最好的那一个，这样就可以被推荐它为整个需求评审方案中的一个构成部分。

RQ3：由 8 种相似度函数和 4 种搜索算法组成的 32 种组合中，哪一种组合在选择一组不相似的 RUCM 用况场景和需求缺陷发现率(DDR)两方面的效能最好?

RQ4：针对每一种相似度函数，RUCM 用场景的平均相似度和这组用况场景的需求缺陷发现率(DDR)之间的关系是怎样的？

在 RUCM 模型的背景下，每一个 RUCM 用况场景都封装了一系列顺序执行的 RUCM 需求描述语句，不同的 RUCM 用况场景可能包含相同的 RUCM 需求描述语句。因此，

本文做如下假设，即，给定 RUCM 用况场景的总数量时，一组平均相似度低的 RUCM 用况场景能够有机会发现更多的需缺陷。这个研究问题就是被设计来解决以上假设。给定 RUCM 用况场景的总量时，通过研究适应度函数值(即代表了平均相似度)和相应的需求缺陷发现率(DDR)之间的相关性，对以上假设进行回答。

RQ5：针对每一种相似度函数和搜索算法的组合，它的功效是怎样的？(其中功效被定义为平均每一个 RUCM 用况场景所包含的需求缺陷数)

在工业实践中，基于 RUCM 的用况评审通常需要在评审成本(即需要进行评审的 RUCM 用况场景的总量)和效能(即这些 RUCM 用况场景所覆盖掉的需求缺陷的总量)进行抉择。最理想的情形就是只选择出那些含有需求缺陷的 RUCM 用况场景参与评审活动。因此，该研究问题被设计来评估每一种相似度函数和搜索算法组合的功效。其中功效计算公式为：$\frac{\#(\text{发现的需求缺陷总数})}{\#(\text{RUCM 用况场景的总量})}$。为了研究 S^3RUCM 的时间性能，本文设计了如下的研究问题 RQ6。

RQ6：针对每一种相似度函数和搜索算法的组合，它是否在可接受的时间内发现 RUCM 用况场景选择方案？

搜索算法的运行时间取决于所要解决的优化问题的规模和特点。该研究问题主要来验证 S^3RUCM 在工业实践中的时间性能。

5.4.3 实验设计

本小节主要阐述实验设计的各个细节。

5.4.3.1 案例描述

本章的实验中，选用了一个工业案例和六个文献案例对 S^3RUCM 进行实验分析。表 24 对这些案例的特性进行了描述。在该实验中，总共选用了 21 个用况并采用 RUCM 进行用况建模和用况规约，这 21 个用况便构成了 21 个用况场景选择的优化问题。在表 24 中，同时给出了产生的 RUCM 用况场景的总数。该实验采用了全条件覆盖”*All Condition Coverage*”(5.2.1 节)的标准进行 RUCM 用况场景的生成，因为这种覆盖标准可以保证每个引起 RUCM 分支的条件都被至少覆盖掉一次。采用哪种覆盖策略产生 RUCM 用况场景并不会影响实验的结果，因为在实验中针对每个优化问题，所有的搜索算法都会采用同样的 RUCM 用况场景集合作为搜索的输入。在表 24 中，针对每个研究案例，同时还给出了 RUCM 用况的平均长度(即 RUCM 基本流中所包含的 RUCM 需求描述语句的平

均数量)和 RUCM 用况场景的平均长度(即 RUCM 用况场景中所封装的 RUCM 需求描述语句的平均数量)。

实验中的工业案例是一个飞行控制系统(NAS)，它对数据采集器返回的各种飞行参数按照控制律计算公式进行数据计算，并产生相应的飞行控制指令，将飞行控制指令发送给飞行舵机从而控制飞行器的安全飞行。该飞行控制系统(NAS)具有两种飞行操作模式：自动驾驶飞行模式，在该模式下不需要飞行员的飞行指令；人工驾驶飞行模式，该模式需要飞行员的飞行指令。一个飞行员可以切换这两种飞行模式。在进行这种综合模块化航电系统 Integrated Modular Avionics (IMA) systems 的开发过程中，我们的工业合作伙伴严格遵循相关的工业标准如 DO-178C[2]，采用模型驱动的技术。这样的系统通常由一系列计算模块、不同类型的硬件设备、系统与执行者、其他系统或外部系统之间复杂的通讯所构成。在这类系统的需求开发过程中，除了系统功能需求外，还有大量的安全需求和异常处理需要被准确捕获并进行精准描述。此外，工业标准 DO-332[3]针对如何使用面向对象技术(Object-Oriented Technology (OOT))及其相关技术进行航空系统的开发给出了详细的指导规则。因此，用况建模技术得到我们工业合作伙伴的青睐并表现出了初步的优势。另外一种基于 RUCM 的变体技术[166]被应用于汽车工业领域并取得初步的研究成果。

实验中的工业案例的 RUCM 用况模型是在工业合作伙伴的协助下共同开发完成的。具体讲，来自工业领域的一位有着四年航空系统开发经验的工程师参与了该实验，他首先被系统化的进行了 RUCM 技术的培训，然后该工程师基于自己的领域知识和工业经验开发了最初的 RUCM 用况模型包含 RUCM 需求规约。为了进一步修缮 RUCM 的用况描述，作者又严格遵循 RUCM 提出的需求描述限制规则对工程师开发的用况描述从遵循 RUCM 限制规则的角度出发进行完善。作者并没有修改原始 RUCM 用况描述的语义，而且每次对 RUCM 用况描述的修改都要经过工程师的确认。对于其他的文献案例，作者采用 RUCM 严格仔细的对其原始的需求进行 RUCM 规约，而不改变它们原始的需求语义。RUCM 需求建模借助 RUCM 工具(http://zen-tools.com/rucm/Editors.html)完成。

表 24 案例的特性描述

Category	*Case Studies*	*# of Use Cases*	*# of Scenarios*	*Average Length of Scenarios*	*Average Length of Use Cases*
Real-world Case Study	Aircraft Navigation System (NAS)	8	736	26	23
Case Studies from	Bank System (ATM)	4	221	20	17

Literature	Crisis Management System (CMS)	1	282	32	60
	Car Part Dealer (CPD)	3	102	27	18
	Video System (VS)	1	36	24	20
	Cab Dispatching System (CDS)	1	34	18	14
	Elevator System (ES)	3	124	29	23
		Total: 21	**Total: 1535**	**Average: 26**	**Average: 25**

为了对 S^{3}RUCM 进行评估，该实验中从 NAS 系统选取了以下 8 个用况进行 RUCM 需求描述。启动系统 Start System (UC1)；上电自检 Power-up Built-in Test (UC2)，主要负责系统上电后的一系列设备、部件检查工作；故障处理 Handle Faults (UC3) 主要被设计来解决系统运行中出现的各种错误和异常；数据采集 Sample Data (UC4)负责从各个传感器采集各种飞行数据；系统同步 Synchronize with SystemB (UC5) 负责实现与另一个对等系统的同步工作，NAS 在设计上采用双机冗余的策略以保证其安全可靠；表决输入数据 Vote InputData (UC6)对收到的传感器采集数据进行验证和处理；表决输出数据 Vote OutputData (UC7)对计算后产生的飞行指令数据进行校验和处理；传输数据 Transmit Data (UC8)将各种表决后的数据传输给对方对等系统。

基于文献调研，本文另外选取了六个其他的文献案例：ATM (Automated Teller Machine)[126]、CMS (Crisis Management System)[53]、CPD (Car Part Dealer system)、VS (Video Store system)、CDS (Cab Dispatching System)和 ES (Elevator System)。具体介绍如下，ATM 是出现在著作[126]中的一个银行系统，本文使用 RUCM 描述了如下四个用况：Withdraw Fund (UC9)、Transfer Fund (UC10)、Query Account (UC11) 和 Validate PIN (UC12)。Withdraw Fund 描述了一个客户如何从银行账户中提取一定数量的钱款的业务流程；Transfer Fund 主要负责如何帮助客户实现在两个合法的银行账户间进行钱款的转账；Query Account 意味着客户收到有关自己银行账户余额的相关信息；Validate PIN 负责对 ATM 客户的 PIN 码进行验证。Capozucca[127]等定义了有关汽车碰撞的危机管理系统软件产品线的一系列需求，即 bCMS-SPL，本文选取了其中的一个用况进行 RUCM 用况描述，即 Communicate with other coordinator (UC13)。CPD、VS、CDS 在文献[8,23]中用于对 RUCM 方法及其实现的从 RUCM 用况模型到 UML 分析模型的模型转换进行评估。在本章的实验中，从 CPD 中选取了三个用况：Create Customer Order (UC14)、Complete Pending Order (UC15)、Order Parts (UC16)；从 VS 中选择了一个用况：Check Database (UC17)；从 CDS 中选取了一个用况：Handle Immediate Job (UC18)。 ES 是一个电梯控制系统，该案例来自加拿大滑铁卢大学的软件需求工程课“SE463 Software Requirements

Specification and Analysis"[3]，本文中选取了三个用况：Turn On (UC19)、Transport Passenger (UC20) 、Change Elevator Work Mode (UC21)。由于这些用况描述来自不同的领域和不同的出处，本章实验中对它们原始的用况描述采用 RUCM 进行了重新描述并未改变它们原始的语义。

5.4.3.2 RUCM 需求故障注入

深深意识到开展需求评审的困难，例如，如何控制评审人员的个人经验、能力的差异对评审结果的影响等，本章的实验中采用故障注入的方式替代传统的手工需求评审的过程对 S^{3}RUCM 进行评估。具体讲，首先针对选取的 RUCM 用况，根据第 4 章提出的 RUCM 变异分析技术创建各种不同的 RUCM 需求缺陷，这样在随后产生的 RUCM 用况场景中就会包含各个需求缺陷；然后，使用 S^{3}RUCM 用况场景选择方法选取相应的用况场景组合；最后，针对 S^{3}RUCM 的每一个实例(即，一个具体的相似度函数与搜索算法的组合)返回的一组 RUCM 用况场景，统计其中包含的 RUCM 需求缺陷的总数量即可对 S^{3}RUCM 进行相应的评估。由于采用故障注入的方式，因此所有的 RUCM 需求缺陷及其总数量都是可以确定的，这样在评价不同的 S^{3}RUCM 的实例时可以通过计算相应的缺陷发现率$DDR = \frac{\#(\textit{发现的需求缺陷总数})}{\#(\textit{手工注入的}RUCM\textit{需求缺陷总量})}$实现。采用这种方式，避免了人工因素对需求评审结果的影响，因此本实验中不需要进行人工需求评审。

本文第 4 章提出一种 RUCM 变异分析的方法，本实验中采用这个方法创建 RUCM 需求缺陷。具体地讲，一个 RUCM 变异体就是使用 RUCM 变异算子对原始的 RUCM 模型进行的一次修改操作，从而可以创建一个特定的 RUCM 需求缺陷。例如，图 33 中，RUCM 备选流'alt1'中有个分支标记语句'RFS 2'用于指明'alt1'是从基本流的第二个语句分支出来的。如果将'RFS 2'修改为'RFS 3'，这将会导致一个需求错误，相应的修改后的 RUCM 规约就是原先的 RUCM 模型的一个变种。RUCM 变异算子是通过严格遵循工业标准 HAZOP[73]对 RUCM 模型中的各个构成要素进行系统分析后得出的。另外，对 RUCM 变异算子的每次使用都会产生一个特定的 RUCM 需求缺陷(例如，模糊性 *Ambiguity*、不正确性 *Incorrectness*)，其中 RUCM 的需求缺陷定义和分类是依据工业标准 IEEE Std. 830-1998[72]和其他的研究成果制定的。

为了系统化的创建 RUCM 变异体，即注入 RUCM 需求缺陷，RUCM 变异分析方法

[3] https://www.student.cs.uwaterloo.ca/~se463/

MuRUCM 给出了一系列的指导规则用于制定相应的缺陷注入策略。下面阐述本实验中具体的故障注入策略：

1) 依据 RUCM 的需求缺陷定义和分类，选择相应的 RUCM 变异算子(Guideline 5)。在该实验中，通过使用相应的 RUCM 变异算子，对所有的 RUCM 需求缺陷分类创建了相应的实例：*Incorrectness* (C1)、*Incompleteness* (C2)、*Inconsistency* (C3)、*Ambiguity* (C4)、*Incomprehensibility* (C5)、*Intestability* (C6)、*Unmodifiability* (C7)、*Infeasibility* (C8)、*Over-Specification* (C9)。

2) 由于该实验中并不知道各个注入的 RUCM 需求缺陷在工业环境中的重要性，即无法获取工业案例的需求开发过程的历史记录，在该实验中并没有区分不同的 RUCM 需求缺陷的重要性(Guideline 1)。例如，给定一个 RUCM 用况规约，假如方案sol_A包含了 3 个 C1 需求缺陷实例，而方案sol_B包含了 1 个 C1 需求缺陷实例和 2 个 C6 需求缺陷实例，那么依据 DDR，sol_A和sol_B的效能是一样的。

3) 面对某种特定的 RUCM 缺陷类型，它可以通过多种 RUCM 变异算子创建相应的实例，该实验中选用那个操作简单的 RUCM 变异算子进行实例创建(Guideline 3 and Guideline 4)。

4) 指导规则 Guideline 2 所示，每一个 RUCM 变异体都对应这一个特定的 RUCM 需求缺陷类型。因此，在使用 RUCM 变异算子时不用考虑消除等效的 RUCM 变异体。

表 25 展示了实验中使用的所有 RUCM 变异算子及其创建的 RUCM 缺陷实例的数量。实验中，每个 RUCM 缺陷都是使用一个相应的 RUCM 变异算子一次而创建的。该实验使用的原始 RUCM 模型都是被确认过的，可以认为其中是不包含需求缺陷的。具体地讲，工业案例的 RUCM 模型是经过工业领域的工程师确认过的；文献案例是借助 RUCM 工具被严格细心重写过的，并没有改变其原始的语义。因此可以认为未知或已存在的需求缺陷的数量为零。另外，该实验中系统化的创建了不同类型的需求缺陷，例如，*Incorrectness* (C1)、*Inconsistency* (C3)、*Ambiguity* (C4)都会涉及语义错误。

如表 25 所示，该实验中通过使用了 30 个不同的 RUCM 变异算子总共创建了 541 个 RUCM 需求缺陷实例。例如，RUCM 变异算子‘DEL-SenBF-C2F2’ 暗示，给定与给 RUCM 用况规约，可以通过删除 RUCM 基本流中的一个 RUCM 需求描述语句从而创建一个不完整性(‘C2F2’)的需求缺陷实例。该实验中，总共创建了 541 个需求缺陷实例并覆盖了 9 种不同的 RUCM 需求缺陷分类：80 个“不正确性”需求缺陷实例、76 个“不完整性”需求缺陷实例、53 个“不一致性”需求缺陷实例；48 个“模糊性”需求缺陷实例、39 个“难

以理解”需求缺陷实例、83 个“不可测”需求缺陷实例、21 个“难以修改”需求缺陷分类、71 个“不可行性”需求缺陷实例、70 个“过度描述”需求缺陷实例。

表 25 采用的 RUCM 变异算子(30)和创建的 RUCM 缺陷实例(541)*

Mutation Operator	#Defects	Mutation Operator	#Defects	Mutation Operator	#Defects	Mutation Operator	#Defects
DEL-SenBF-C1F1 (MO1)	21	DEL-SenBF-C2F2 (MO6)	41	ADD-RFSflow-C3F2 (MO10)	18	REP-AS-C4UCS1 (MO14)	13
SWAP-SenBF-C1F3 (MO2)	6	DEL-AF-C2F1 (MO7)	19	DEL-SenBF-C3F1 (MO11)	11	REP-AS-C4F1 (MO15)	16
ICR-RFSsi-C1F5 (MO3)	24	DEL-RFS-C2F3 (MO8)	7	REP-AF-C3F2 (MO12)	12	REP-AS-C4F2 (MO16)	19
ICR-RESsi-C1F6 (MO4)	16	DEL-RES-C2F5 (MO9)	9	ICR-RFSsi-C3F3 (MO13)	12	*Ambiguity*: 48	
DEC-RESsi-C1F6 (MO5)	13	*Incompleteness*: 76		*Inconsistency*: 53		REP-AS-C5F1 (MO17)	18
Incorrectness: 80		SWAP-SenBF-C8F1 (MO23)	12	ADD-SenBF-C9F2 (MO24)	21	REP-PostC-C5F2 (MO18)	21
SWAP-SenBF-C6F3 (MO19)	21	ADD-IFELSEcs-C8F1 (MO24)	46	ADD-SenAF-C9F2 (MO28)	9	*Incomprehensibility*: 39	
DEL-RFS-C6F1 (MO20)	12	DEL-IFELSEas-C8F1 (MO25)	13	ADD-DOas-C9F2 (MO29)	9		
DEC-RFSsi-C6F1 (MO21)	32	*Infeasibility:* 71		ADD-INC-C9H4 (MO30)	19		
REP-toABT-C6F3 (MO22)	18	ADD-SenBF-C9F2 (MO26)	21	ADD-AF-C9F1 (MO31)	12		
Intestability: 83		*Unmodifiability*: 21		*Over-Specification*: 70			

* 变异算子‘ADD-SenBF-C9F2’ 创建了两个不同的缺陷实例：*Unmodifiability* and *Over-Specification*.

5.4.3.3 设计变量

为了系统的评估 S^{3}RUCM 的成本-效益，本节定义了一系列的实验设计变量(独立变量、依赖变量)，表 26 展示了所有的实验设计变量。这些实验设计变量同时也跟 5.4.2 节中设计的研究问题进行了相应的映射。

独立变量。如表 26 所示，在该实验中总共有 4 个直接的独立变量： SIM、ALGO、NUCS、NMTS。SIM 代表了 8 种相似度计算函数：CNT、JAC、GOW、SOK、NLCS、LEV、NW、SW (见 5.2.2 节)。ALGO 代表了采用的 4 种搜索算法：SSGA、(1+1) EA、AVM、RS (见 5.2.3 节)。给定一个 RUCM 用况规约，NUCS 表示需要从产生的 RUCM 用况场景中选择出多少个，在该实验的设计中，NUCS 采用总数量的百分比，即 NUCS 的取值是从 10%到 90%并按照 10%递增。NUCS 暗含了进行需求评审的工作量，即 S^{3}RUCM 的成本。NMTS 是注入的 RUCM 需求缺陷实例的总量。基于 SIM 和 ALGO，又创建了一个间接独立变量 CSA，即 8 种相似度函数和 4 种搜索算法构成的 32 种组合。另外，本文采用$CSA_{SIM-ALGO}$表示一个特定的 CSA 实例。例如，$CSA_{NW-(1+1)\,EA}$表示 NW 相似度函数和(1+1) EA 搜索算法的组合。

表 26 实验设计变量(独立变量，依赖变量)

变量类型	变量名称	变量取值	与研究问题RQs的映射
独立变量	*SIM*	*CNT, JAC, GOW, SOK, NLCS, LEV, NW, SW*	RQ1, RQ2, RQ3
	ALGO	*SSGA, (1+1) EA, AVM, RS*	
	NUCS	10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%	RQ1, RQ2, RQ3, RQ4, RQ5, RQ6

	NMTS	给定一个RUCM用况规约，注入的RUCM缺陷故障的总数	RQ1, RQ2, RQ3, RQ4, RQ5
	CSA	SIM和ALGO构成的32种组合	RQ4, RQ5, RQ6
依赖变量	*FV*	实数	RQ1, RQ2, RQ3, RQ4
	DDR	实数	RQ1, RQ2, RQ3, RQ4, RQ5
	EFF	实数	RQ5
	RT	实数	RQ6

依赖变量。第一个依赖变量是适应度函数值 fitness value (FV)。在该实验中，给定一个搜索优化问题，每一个搜索算法都会在完成 25000 次搜索比较后返回一个最好的方案，然后对这个方案进行适应度函数计算就返回了 FV。给定一个 RUCM 用况规约ucs，即搜索优化问题，FV 的计算过程如下：

$$FV_{(NUCS,\ CSA)} = \min_{1\le k\le 25000}(Fitness(sol_i)) \quad (16)$$

其中$Fitness(sol_i)$是公式(15)表示的适应度函数(见 5.3.2 节)，它主要依据当前 CSA 中采用的相似度函数来计算方案sol_i返回的那组 RUCM 用况场景的平均相似度并进行相应的适应度函数转换。FV 是当前 CSA 中采用的那个搜索算法在经过 25,000 比较后返回的最小值。 在该实验中可以使用 FV 表示平均相似度(由公式(12)进行计算)对 CSA 的效能进行评估。

另一个依赖变量是缺陷发现率 DDR。给定一个ucs和CSA实例，DDR 表示当前CSA实例发现的所有缺陷实例的数量和注入的全部缺陷实例数量的比值：

$$DDR_{(CSA,\ NMTS)} = \frac{The\ number\ of\ detected\ RUCM\ defects\ of\ a\ particular\ CSA}{NMTS} \quad (17)$$

给定一个方案sol_i，功效变量 EFF 用来计算sol_i中每个 RUCM 用况场景平均能发现多少个 RUCM 需求故障实例：

$$EFF_{(CSA,\ NUCS)} = \frac{The\ number\ of\ detected\ RUCM\ defects\ of\ a\ particular\ CSA}{NUCS * the\ total\ number\ of\ generated\ scenarios} \quad (18)$$

最后，运行时间变量 RT 用来计算每个 CSA 实例用以解决所有优化问题的平均运行时间：

$$RT_{CSA} = \frac{\sum_{i=1}^{NR_{problem}} Time_{(CSA,\ problem_i)}}{NR_{problem} * 100} \quad (19)$$

每一个 RUCM 用况规约都会产生一组 RUCM 用况场景，而选择出一组用况场景以发现尽可能多的 RUCM 需求缺陷就构成了一个优化问题。$NR_{problem}$就代表了所有的 RUCM 用况规约，而$problem_i$则指明了第 i 个优化问题。在该实验中$NR_{problem} = 21$，因为总共有 21 个 RUCM 用况规约(5.4.3.1 节中表 24)。$Time_{(CSA,\ problem_i)}$表示一个特定的 CSA 实例运行 100 次后的总消耗时间。为尽量减少搜索算法的随机影响，在本实验

中每个搜索算法都会执行 100 次，并把这 100 次的返回结果作为统计计算的样本，有关搜索算法的详细设置参阅 5.4.3.6 节。

5.4.3.4 统计方法

为了对实验获得的结果数据进行科学分析，本文依据文献[131]给出的指导规则，使用 Vargha-Delaney[77]统计方法、Kruskal-Wallis[78]秩检验、Wilcoxon[79]秩和检验、Bonferroni[129]修正方法、Kendall's-Tau-test[167]检验进行统计分析并对结果进行汇报。如同文献[168]中所倡导的，该实验首先使用 Shapiro-Wilk[168]检验对实验结果数据的正态性进行检测用以选择合适的统计方法。实验中将显著水平设置为 0.05，即一个样本服从正态分布当且仅当 Shapiro-Wilk 检验的p-value 大于 0.05。实验中使用 Shapiro-Wilk 检验对所有的实验结果数据进行检测，依据 Shapiro-Wilk 检验的结果发现该实验的所有结果数据都不服从正态分布，因此在本实验中依据文献[131]中的指导规则选用以上非参统计分析方法。本文中所有的那些由 Wilcoxon 秩和检验返回的p-value 都已使用 Bonferroni 修正方法进行了修正。

在该实验中，首先使用 Kruskal-Wallis 秩和检验对多组样本数据进行统计分析，从而确定在这多组结果数据中是否存在显著差异。例如，为了研究 S^{3}RUCM 中采用的搜索算法的效能(即 RQ2)，在实验中首先采用 Kruskal-Wallis 秩和检验对所有算法返回的数据进行检验。如果 Kruskal-Wallis 秩和检验的结果显示存在一个显著差异(即p-values 小于 0.05)，然后再对每一对搜索算法进行成对比较以分析具体差异。

其次，在实验中采用 Vargha-Delaney 效应因子、Wilcoxon 秩和检验和 Bonferroni[129]p-value 修正方法进行多重比较。其中，Vargha-Delaney 统计方法用于计算$\hat{A}_{12}$，$\hat{A}_{12}$是一个非参的效应因子。在该实验的语境下，给定一个相似度函数，$\hat{A}_{12}$用于比较两个搜索算法 A 和 B 谁更有可能返回一个数值小的适应度函数值 FV。如果$\hat{A}_{12}$是 0.5，那么这两个算法有相同的效能。如果$\hat{A}_{12}$大于 0.5，那么 A 比 B 有更高的几率返回一个好的方案，反之则 B 比 A 有更高的概率返回一个好的方案。 Wilcoxon 秩和检验搭配 Bonferroni 修正方法用于计算p-value 并决定 A 和 B 的返回结果之间是否存在着显著差异。在该实验的统计分析中，将显著水平设定为 0.05，即，如果p-value 小于 0.05 则说明存在显著差异。

给定一个相似度函数，针对所有搜索算法返回的优化方案sol_i，为了研究 RUCM 用况场景组对应的平均相似度 FV 和其需求缺陷发现率 DDR 之间的相关性，该实验中采用 Kendall's Tau (τ) test。Kendall 检验的τ值的取值范围是[-1, 1]。如果τ值趋向于 1 则两

组数据之间存在正相关性；如果τ值趋向于-1 则两组数据之间存在负相关性；如果τ值趋向于 0 则两组数据之间不存在相关性。此外，该实验中同时也对相关性检验的显著水平p-value 进行报告，如果p-value 小于 0.05 则当前的 Kendall 检验结果是显著的。

表 27 细化的研究问题及相应的统计检验*

RQ	*Refined Research Question*	*Dependent Variable*	*Statistical Test*
RQ1	T1.1: 针对每一个相似度函数，比较每一个搜索算法与 *RS 的 FV*	*FV*	Kruskal-Wallis rank test Vargha and Delaney statistics Wilcoxon rank sum test in conjunction with Bonferroni correction
	T1.2: 针对每一个相似度函数，比较每一个搜索算法与 *RS 的 DDR*	*DDR*	
RQ2	T2.1: 针对每一个相似度函数，比较每一对搜索算法的 *FV*	*FV*	
	T2.2: 针对每一个相似度函数，比较每一对搜索算法的 *DDR*	*DDR*	
RQ3	T3: 比较每一对 *CSA* 的 *DDR*	*DDR*	
RQ4	T4: 针对每一个相似度函数，使用所有算法返回的 *FV* 和 *DDR*，并计算 *FV* 和 *DDR* 之间的相关性	*FV, DDR*	Kendall Tau test
RQ5	T5: 对每一个 *CSA* 实例评估其 *EFF*	*EFF*	N/A
RQ6	T6: 对每一个 *CSA* 实例计算其平均运行时间	*RT* (in seconds)	N/A

*N/A: 不需要进行统计检验

为了回答 5.4.2 节中设计的研究问题 RQs，如表 27 所示，本实验设计了一系列的检验工作。两个检验工作 T1.1 和 T1.2 被设计来回答 RQ1。为回答 RQ2，T2.1 和 T2.2 分别从 FV 和 DDR 两方面对 CSA 进行评估。T3 被设计来确定 FV 和 DDR 的相关性以回答 RQ3。T4 被设计来评估不同的 CSA 实例关于 DDR 的效能以回答 RQ4。为回答 RQ5，T5 用来评估 CSA 的功效。为回答 RQ6，T6 用来对每个 CSA 实例进行平均执行时间的计算。

5.4.3.5 统计假设

表 28 总结了针对 RQ1-RQ4 设计的统计假设检验。如表 28 所示，针对 RQ1 本文设计了如下两种零假设。H_{01}：给定一个相似度函数SIM，关于适应度函数值 FV，在任意一个搜索算法与随机搜索算法 RS 之间不存在显著性差异，例如，$FV_{SIM}(SSGA) = FV_{SIM}(RS)$；$H_{02}$：给定一个相似度函数$SIM$，关于需求缺陷发现率 DDR，在任意一个搜索算法与随机搜索算法 RS 之间不存在显著性差异，例如，$DDR_{SIM}(AVM) = DDR_{SIM}(RS)$。

针对 RQ2，本文设计了如下两种零假设。H_{01}：给定一个相似度函数SIM，关于适应度函数值 FV，除随机搜索 RS 外，任意一对搜索算法之间不存在显著性差异，如，$FV_{SIM}(SSGA) = FV_{SIM}(AVM)$；$H_{02}$：给定一个相似度函数$SIM$，关于需求缺陷发现率 DDR，除随机搜索 RS 外，任意一对搜索算法之间不存在显著性差异，如，$DDR_{SIM}(SSGA) = DDR_{SIM}((1+1)EA)$。

针对 RQ3，其统计零假设如下。H_{01}：就需求缺陷发现率 DDR 而言，任意一对 CSA 实例之间不存显著性差异，即$DDR(CSA_i) = DDR(CSA_j)$，其中CSA_i和CSA_j是相似度函数和搜索算法构成的 32 个组合中的任意一对。

针对 RQ4，其统计零假设如下。H_{01}：给定一个相似度函数SIM，适应度函数值 FV 与缺陷发现率 DDR 之间不存在显著的相关性，即$Tau_{SIM}(FV, DDR) = 0$。

表 28 RQ1-RQ4 的假设检验

RQ	*Null Hypothesis*	*Alternative Hypothesis*
RQ1	$H_{011}: FV_{SIM}(SSGA) = FV_{SIM}(RS)$	$H_{111}: FV_{SIM}(SSGA) \neq FV_{SIM}(RS)$
	$H_{012}: FV_{SIM}((1+1)EA) = FV_{SIM}(RS)$	$H_{112}: FV_{SIM}((1+1)EA) \neq FV_{SIM}(RS)$
	$H_{013}: FV_{SIM}(AVM) = FV_{SIM}(RS)$	$H_{113}: FV_{SIM}(AVM) \neq FV_{SIM}(RS)$
	$H_{021}: DDR_{SIM}(SSGA) = DDR_{SIM}(RS)$	$H_{121}: DDR_{SIM}(SSGA) \neq DDR_{SIM}(RS)$
	$H_{022}: DDR_{SIM}((1+1)EA) = DDR_{SIM}(RS)$	$H_{122}: DDR_{SIM}((1+1)EA) \neq DDR_{SIM}(RS)$
	$H_{023}: DDR_{SIM}(AVM) = DDR_{SIM}(RS)$	$H_{123}: DDR_{SIM}(AVM) \neq DDR_{SIM}(RS)$
RQ2	$H_{011}: FV_{SIM}(AVM) = FV_{SIM}(SSGA)$	$H_{111}: FV_{SIM}(AVM) \neq FV_{SIM}(SSGA)$
	$H_{012}: FV_{SIM}(AVM) = FV_{SIM}((1+1)EA)$	$H_{112}: FV_{SIM}(AVM) \neq FV_{SIM}((1+1)EA)$
	$H_{013}: FV_{SIM}(SSGA) = FV_{SIM}((1+1)EA)$	$H_{113}: FV_{SIM}(SSGA) \neq FV_{SIM}((1+1)EA)$
	$H_{021}: DDR_{SIM}(AVM) = DDR_{SIM}(SSGA)$	$H_{121}: DDR_{SIM}(AVM) \neq DDR_{SIM}(SSGA)$
	$H_{022}: DDR_{SIM}(AVM) = DDR_{SIM}((1+1)EA)$	$H_{122}: DDR_{SIM}(AVM) \neq DDR_{SIM}((1+1)EA)$
	$H_{023}: DDR_{SIM}(SSGA) = DDR_{SIM}((1+1)EA)$	$H_{123}: DDR_{SIM}(SSGA) \neq DDR_{SIM}((1+1)EA)$
RQ3	$H_{011}: DDR(CSA_I) = DDR(CSA_J)$	$H_{111}: DDR(CSA_I) \neq DDR(CSA_J)$
RQ4	$H_{011}: Tau_{SIM}(FV, DDR) = 0$	$H_{111}: Tau_{SIM}(FV, DDR) \neq 0$

5.4.3.6 参数配置

在基于搜索的软件工程 SBSE 实践中，搜索算法被用来解决各种优化问题[151]，其中开源框架 jMetal[169]是 SBSE 研究中使用最广泛的工具。截止到本文进行实验设计时，jMetal 已经收录并实现了 13 种多目标优化搜索算法用于解决多目标优化问题，3 种不同的策略用于解决单目标优化问题。由于本文中的 RUCM 用况场景选择问题是一个单目标的优化问题，该实验中选择了相应的代表性搜索算法。SSGA 作为遗传算法的代表被选用，(1+1) EA 是进化算法的代表，SSGA 和(1+1) EA 都是全局优化技术，AVM 则代表局部优化算法。该实验中选用 SSGA 而不用 GGA 是因为 SSGA 使用更少的计算成本(即每一次迭代的计算代价)却有更好的表现[158]。已有的研究[149,153]证明上述被选用的搜索算法在解决单目标的工程优化问题都有非常好的表现。将选用的搜索算法与随机搜索 RS 进行对比在 SBSE 的研究中也是非常重要的，因为这样做才能保证所要解决的工程优化问题不是一个朴素的问题，需要采用更有效的搜索算法进行解决。因此，将 RS 作为基线并设计实验进行对比分析是 SBSE 研究中的一种通用做法[163]。

本实验中各个搜索算法的参数设置如下。针对 SSGA，种群的大小设置为 100，同时采用排序选择算子[160]对个体进行选择。每一对父母采用标准的单点交叉策略创建相应的孩子方案，其中交叉的概率设定为 0.9。每个元素变量的变异概率都是一样的，即$1/n$，其中n是当前个体中所有元素变量的总数。已有的研究[164,170]中(1+1) EA 表现的要比 GAs 好，因此本文实验中选用了(1+1) EA。针对(1+1) EA，采用同样的变异概率，即$1/n$。RS 在本文的实验中被用作基线[162]以比对其他的搜索算法。对搜索算法不同的参数配置可能会引起搜索算法不同的表现，但是在实践中经常采用标准的参数配置[171]。此外，本实验中所有搜索算法都要进行 25000 次的比较，即每个返回的方案sol_i都是在搜索算法进行了 25000 次比较后返回的适应度函数值 FV 最小的那一个。

相似度函数最初是被广泛应用于生物信息学的研究中[141,142]，现在它们被引入到 SBSE 中用以解决测试用例选择和优化的问题。其中，以下八种相似度函数：HD、CNT、JAC、GOW、SOK、LEV、NW、SW 是 SBSE 中最为广泛使用的[149]。本文选用的相似度函数中，NW 和 SW 需要进行相应的参数设置，即元素匹配的权限、元素不匹配的权限、元素空缺的权限。在本文的实验设计中采用 Needleman 和 Wunsch 发表的研究工作[172]中的参数配置，即元素匹配的权限设置为+1、元素不匹配的权限设置为-1、元素空缺的权限设置为-1。

5.4.4 实验执行

根据文献[131]提供的指导规则，每个搜索算法执行 100 次以克服算法本身的随机影响。 给定一组 RUCM 用况场景SN_{ucs}(由 RUCM 用况规约ucs自动生成)，搜索算法以SN_{ucs}为输入，在整个搜索过程中 S^{3}RUCM 会针对每个 NUCS 值 (即 10%，20%，30%，40%，50%，60%，70%，80%，90%)对每一个 CSA 实例(总共 32 个)执行 100 次独立的搜索。在每一次搜索中，搜索算法都要经过 25000 次的比较才返回适应度函数值最小的那个方案sol_i。整个实验的具体执行是在计算机集群 Abel[4]上实施的，Abel 为每个计算用户提供 8 个计算节点，每个计算节点由 16 个物理计算内核和 64GB 的内存构成。

针对实验中的每一个 RUCM 用况规约ucs，需要创建 9 个不同的实验结果数据集，即$D_1 \ldots D_9$，其中D_i对应于一个具体的 NUCS 值(即 10%，20%，30%，40%，50%，60%，70%，80%，90%)的所有 32 个 CSA 实例返回的实验结果数据。具体地讲，由于 S^{3}RUCM

[4] The Abel cluster: http://www.uio.no/english/services/it/research/hpc/abel/

方法中采用了 32 中不同的组合(即 8 种相似度函数和 4 种搜索算法)进行搜索选择，每个D_i实际上是由 32 不同的数据集构成，即 $D_i = \{D_{cm_1} \dots D_{cm_{32}}\}$其中$D_{cm_i}$是某个具体的 CSA 实例的返回数据。更进一步，$D_{cm_i}$包含了经过 100 次独立运行后返回的 100 个独立的优化方案sol_i以及每个优化方案所对应的经过 25000 次比较后的最小的适应度函数值 FV。此外，每个 CSA 实例的运行时间也被记录收集。

5.4.5 结果分析

本节针对每个研究问题 RQ 的实验结果进行分析，详细的实验结果数据可以参加附录。为了方便本实验的重现，相关的实验说明和实验材料可以参考公开网站 http://zen-tools.com/rucm/S3RUCM.html。

5.4.5.1 RQ1 的实验结果及分析

RQ1 主要的研究目标是分别从 *FV* 和 *DDR* 两个角度对选择的搜索算法与随机搜索 RS 进行对比分析。表 29 展示了实验中一个优化问题的数据：从 RUCM 用况规约‘*Query Account*’ 的全部用况场景中选择 10%。完整的 RQ1 结果数据在附录表 1 中进行了报告。

给定一个相似度函数 SIM，针对搜索算法返回的 *FV* 和 *DDR*，首先使用 Kruskal–Wallis 检验 进行统计分析。如果相应的p-value 小于 0.05，然后再针对每一个搜索算法与 RS 进行成对比较。在进行成对比较时，Vargha-Delaney 统计方法用于确定任意一个搜索算法与 RS 之间是否存在差异，Wilcoxon 秩和检验搭配 Bonferroni 修正方法用来判定这个差异是否显著。例如，在表 29 中，针对相似度函数 CNT，SSGA 表现的要比 RS 显著得好，不论是针对 FV 的比较还是针对 *DDR* 的比较。因为针对 *FV*，$\hat{A}_{12}$= 1 且p-value < 0.05；针对 *DDR*，$\hat{A}_{12}$= 0.643 且p-value < 0.05。

为了对附录表 1 中 RQ1 的实验数据进行详细分析，本节依据附录表 1 中的数据创建了图 34 和图 35 进行分析说明。针对每一个相似度函数，分别从 *FV* 和 *DDR* 出发， 对每一个搜索算法与随机搜索 RS 进行成对比较，统计它们在解决所有的优化问题中有显著性表现的次数。对于一个搜索算法$ALGO_i$，OptimumPercentage 定义为该搜索算法$ALGO_i$在所有的优化问题中，比 RS 显著性好地解决了优化问题的总数与全部优化问题总数的比值。图 34 展示了针对 *FV* 的 OptimumPercentage 的一个实例， 图 35 展示了针

对 *DDR* 的 OptimumPercentage 的一个实例。

表 29 RQ1 的部分实验结果

SIM		*SSGA vs. RS*		*AVM vs. RS*		*(1+1) EA vs. RS*	
		$\hat{A}_{12}$	*p-value*	$\hat{A}_{12}$	*p-value*	$\hat{A}_{12}$	*p-value*
CNT	***FV***	1	< 0.05	0.9979	< 0.05	1	< 0.05
	DDR	0.643	< 0.05	0.70955	< 0.05	0.6391	< 0.05
JAC	***FV***	1	< 0.05	1	< 0.05	1	< 0.05
	DDR	0.7438	< 0.05	0.6957	< 0.05	0.75585	< 0.05
GOW	***FV***	1	< 0.05	0.9958"	< 0.05	1	< 0.05
	DDR	0.63885	< 0.05	0.71035	< 0.05	0.65875	< 0.05
SOK	***FV***	1	< 0.05	0.9997	< 0.05	1	< 0.05
	DDR	0.8192	< 0.05	0.69205	< 0.05	0.74175	< 0.05
NLCS	***FV***	1	< 0.05	0.9978	< 0.05	1	< 0.05
	DDR	0.64495	< 0.05	0.6929	< 0.05	0.67585	< 0.05
LEV	***FV***	1	< 0.05	0.99735	< 0.05	1	< 0.05
	DDR	0.50255	0.8186	0.62185	< 0.05	0.5538	0.1087
NW	***FV***	1	< 0.05	1	< 0.05	1	< 0.05
	DDR	0.9206	< 0.05	0.72905	< 0.05	0.88615	< 0.05
SW	***FV***	1	< 0.05	0.999	< 0.05	1	< 0.05
	DDR	0.9464	< 0.05	0.743	< 0.05	0.90245	< 0.05

*使用 Vargha-Delaney 统计，Wilcoxon 秩和检验和 Bonferroni 修正方法，显著水平为 0.05 (RUCM 用况规约来自 ATM 案例中的‘*Query Account*’，同时 *NUCS* = 10%)

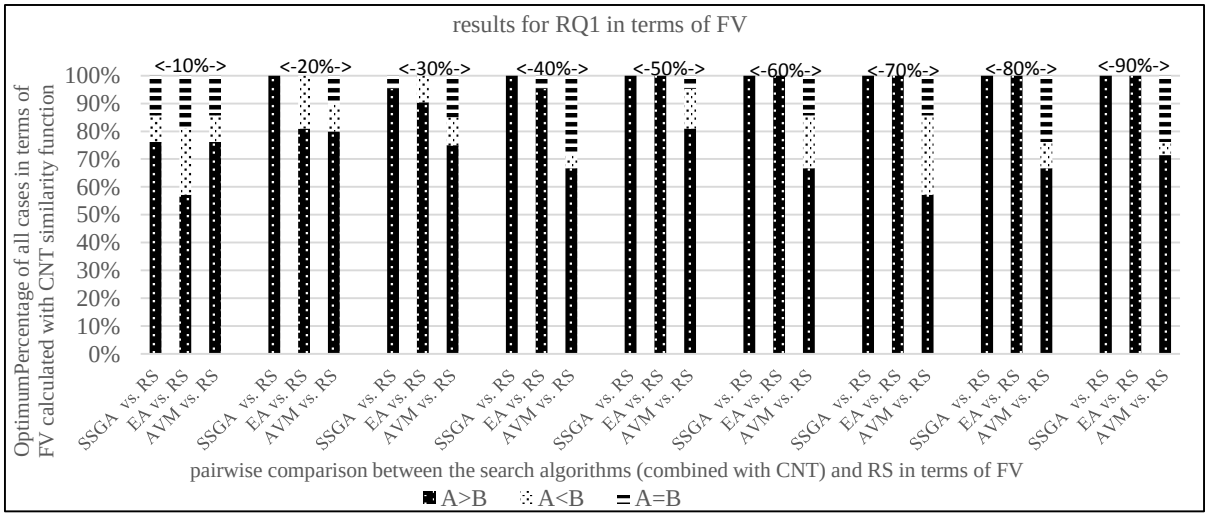

图 34 针对 FV，将选择的任意搜索算法与 RS 进行成对比较分析(基于附录表 1 的数据创建) *

*EA: (1+1) EA

针对 *FV*，各个搜索算法与 RS 的成对比较。***SSGA vs. RS***，在 RUCM 用况场景选择的优化问题的解决中，*SSGA* 要比 *RS* 显著得好，无论使用哪种相似度函数还是 *NUCS* 取何值。以相似度函数 *CNT* 为例，如图 34 所示，当 *NUCS* = 10%，*SSGA* 与 RS 相比，能够显著性好地解决 76.19%的所有优化问题(即 16 个优化问题，在数据表中由‘A>B’所示)；而 *RS* 显著性好的比 *SSGA* 解决了 14%的所有优化问题(即 3 个优化问题，在数据表中由‘A<B’所示)；在剩余的两个问题中，*SSGA* 与 *RS* 之间并没有显著性的差异(在数据表中

由‘A=B’所示)。相似的结果趋势也在其他的 *NUCS* 取值(20%、30%、40%...90%)中可以观察到。***(1+1)EA vs. RS***，在 RUCM 用况场景选择的优化问题的解决中，*(1+1)EA* 要比 *RS* 显著得好，无论使用哪种相似度函数还是 *NUCS* 取何值。如图 34 所示，当 *NUCS* = 30%，*(1+1)EA* 与 *RS* 相比，能够显著性好地解决 90.19%的所有优化问题(即 19 个优化问题，在数据表中由‘A>B’所示)，而 *RS* 只是显著性好地解决了其中的 2 个优化问题(在数据表中由‘A<B’所示)。相似的结果趋势也在其他的 *NUCS* 取值(20%、30%、40%...90%)中可以观察到。***AVM vs. RS***，*AVM* 要比 *RS* 显著得好。以 *NUCS*=20% 为例，*AVM* 与 RS 相比，能够显著性好地解决 80.95%的所有优化问题(即 17 个优化问题，在数据表中由‘A>B’所示)；*RS* 显著性好地解决了 2 个优化问题；在剩余的 2 优化问题中 *AVM* 与 RS 并没有显著性的差异。相似的结果趋势也在其他的 *NUCS* 取值(20%、30%、40%...90%)中可以观察到。

针对 *DDR*，各个搜索算法与 *RS* 的成对比较。***SSGA vs. RS***，在 RUCM 用况场景选择的优化问题的解决中，*SSGA* 要比 *RS* 显著得好，无论使用哪种相似度函数还是 *NUCS* 取何值。以相似度函数 *CNT* 为例，如图 35 所示，当 *NUCS* = 30%，*SSGA* 与 *RS* 相比，能够显著性好地解决 90.19%的所有优化问题(即 19 个优化问题，在数据表中由‘A>B’所示)；在剩余的两个问题中，*SSGA* 与 *RS* 之间并没有显著性的差异(在数据表中由‘A=B’所示)。相似的结果趋势也在其他的 *NUCS* 取值(20%、30%、40%...90%)中可以观察到。***(1+1)EA vs. RS***，在 RUCM 用况场景选择的优化问题的解决中，*(1+1)EA* 要比 *RS* 显著得好，无论使用哪种相似度函数还是 *NUCS* 取何值。如图 35 所示，当 *NUCS* = 10%，*(1+1)EA* 与 *RS* 相比，能够显著性好地解决 80.95%的所有优化问题(即 17 个优化问题，在数据表中由‘A>B’所示)；*RS* 仅仅显著性好地解决了 1 个优化问题；在剩余的 3 个优化问题解决中，*(1+1)EA* 与 *RS* 之间并没有显著性差异。***AVM vs. RS***，类似的，我们也可以观察到针对大多数优化问题 *AVM* 表现都要比 *RS* 显著得好，无论 *NUCS* 取何值(10%、20%、30%、40%...90%)。

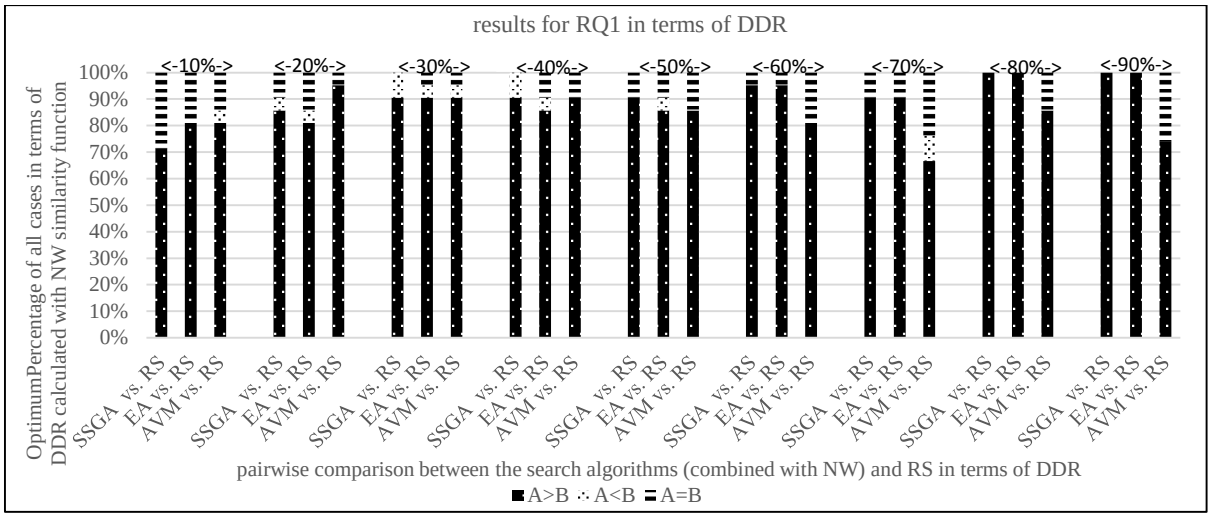

图 35 针对 DDR，将选用的搜索算法与 RS 进行成对比较的结果(基于附录表 1 的数据进行创建) *

*EA: (1+1) EA

基于统计分析的结果数据，可以得出以下结论：对于任何一种相似度函数(*CNT* 、*JAC*、 *GOW*、*SOK*、*NLCS*、*LEV*、*NW*、*SW*)，无论是以 *FV* 为评价指标还是以 *DDR* 为评价指标，搜索算法 *SSGA*、*(1+1)EA*、*AVM* 都比 *RS* 显著得好，无论 *NUCS* 取何值(10%、20%、30%、40%...90%)。即 *SSGA*、*(1+1)EA*、*AVM* 与 *RS* 相比，可以选择一组 RUCM 用况场景，其平均相似度 *FV* 要小同时能够获取更高的需求缺陷检测率 *DDR*。

5.4.5.2 RQ2 的实验结果及分析

为了回答研究问题 RQ2，分别以 *DDR* 和 *FV* 为评价指标，对 *SSGA*、*(1+1)EA*、*AVM* 中的每一对搜索算法进行比较分析。具体地讲，给定一个优化问题(即 RUCM 用况规约 ucs)，针对每一个相似度函数，首先对所有搜索算法的结果数据进行 Kruskal–Wallis 统计分析，如果 Kruskal–Wallis 统计检测返回的p-value 小于 0.05，再使用 Vargha-Delaney 统计、Wilcoxon 秩和检验并搭配 Bonferroni 修正方法对 *SSGA*、*(1+1)EA*、*AVM* 中的每一对搜索算法进行成对分析。其中 Bonferroni 修正方法用于在多重比较中对p-value 进行修正。完整的统计分析结果在附录表 2 和附录表 3 进行报告，本节主要对统计结果进行分析。

针对 *DDR*，表 30、图 36-图 43 依据附录表 2 的统计分析结果进行创建并对所有的成对搜索算法的统计比较结果进行总结。如表 30 所示，搜索算法的表现依据 *NUCS* 的不同取值而变化，下面进行详细的阐述。

表 30 RQ2 关于 *DDR* 的统计比较结果的总结*

SIM	***NUCS***								
	10%	***20%***	***30%***	***40%***	***50%***	***60%***	***70%***	***80%***	***90%***
CNT	M/G/E	M/E>G	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M
JAC	G/M/E	G>E/M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M
GOW	M>G>E	M/G/E	E>G>M	E>G>M	E/G>M	E>G>M	E>G>M	E>G>M	E>G>M
SOK	M/G/E	G>M>E	G>E>M	E/G>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M
NLCS	M/G/E	M/G/E	M/E>G	E>G>M	E/G>M	E>G>M	E>G>M	E>G>M	E>G>M
LEV	G>M/E	G>M>E	M>G>E	E>G>M	E>G>M	E>G>M	G>E>M	E>G>M	E>G>M
NW	G>E>M	G>E>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M	E/G>M
SW	G>M/E	G>E>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M	E>G>M	E/G>M

*G: *SSGA*, E: *(1+1) EA*, M: *AVM*; >: 显著性的优于, /: 差异性不显著

NUCS = 10%。对相似度函数 *CNT* (图 36)、*JAC* (图 37)、*SOK* (图 39)、*NLCS* (图 40)，搜索算法 *SSGA*、*(1+1)EA*、*AVM* 它们在将近一半的优化问题的解决中并没有显著性的差异。以 *JAC* 为例， 如图 37 所示，*SSGA* 与 *AVM* 相比，*SSGA* 能够显著性好地解决 29%的所有优化问题，而 *AVM* 能够显著性好地解决 24%的所有优化问题，在剩余的 48%的优化问题 *SSGA* 与 *AVM* 之间并不存在显著性的差异。对相似度函数 *GOW*，如图 38 所示，*AVM* 取得了最好的统计分析结果，然后是 *SSGA*， *(1+1)EA* 的统计分析结果最差。对相似度函数 *LEV*(图 41)、SW(图 43)，*SSGA* 取得最好的统计分析结果，而*(1+1)EA* 和 *AVM* 之间的统计分析差异并不显著。对相似度函数 *NW*(图 42)，*SSGA* 取得最好的统计分析结果，然后是*(1+1)EA*，*AVM* 的统计分析结果最差。

NUCS = 20%。对相似度函数 *GOW*(图 38)、*NLCS*(图 40)，搜索算法 *SSGA*、*(1+1)EA*、*AVM* 它们之间的统计分析差异并不显著。对相似度函数 NW (图 42)和 SW (图 43)，*SSGA* 取得最好的统计分析结果，然后是 *(1+1)EA*，*AVM* 的统计分析结果最差。对相似度函数 *CNT* (图 36)，*AVM* 和*(1+1)EA* 都要比 SSGA 显著得好，*AVM* 和*(1+1)EA* 之间的统计分析差异并不显著。对相似度函数 *JAC* (图 37)，*SSGA* 取得最好的统计分析结果，而 *AVM* 和 *(1+1)EA* 之间的统计分析差异并不显著。对相似度函数 *SOK* (图 39)、*LEV*(图 41)， *SSGA* 取得最好的统计分析结果，然后是 *AVM*，*(1+1)EA* 的统计分析结果最差。

NUCS = 30%。对相似度函数 *CNT* (图 36)、*JAC* (图 37)、*GOW* (图 38)、*NW* (图 42) 和 SW (图 43)，*(1+1)EA* 取得最好的统计分析结果，然后是 *SSGA*，*AVM* 的统计分析结果最差。对相似度函数 *SOK*(图 39)，*SSGA* 取得最好的统计分析结果，然后是*(1+1)EA*，*AVM* 的统计分析结果最差。对相似度函数 *NLCS* (图 40)，*AVM* 和*(1+1)EA* 要比 SSGA 显著得好，而*(1+1)EA* 和 *AVM* 之间的统计分析差异并不显著。对相似度函数 *LEV* (图 41)，*AVM* 取得最好的统计分析结果，然后是 *SSGA*，*(1+1)EA* 的统计分析结果最差。

NUCS = 40%, 50%, 60%, 70%, 80%, 90%。对绝大多数优化问题，*(1+1)EA* 取得最好

的统计分析结果，而 *AVM* 的统计分析结果最差。具体地讲，对 *SOK* 且 *NUCS* = 40% (图 39)、*GOW* 且 *NUCS* = 50%(图 38)、*NLCS* 且 *NUCS* = 50% (图 40)、*NW* 且 *NUCS* = 90%(图 42)、SW 且 NUCS = 90% (图 43)，*(1+1)EA* 和 *SSGA* 要显著性比 *AVM* 好，*(1+1)EA* 和 *SSGA* 之间的统计分析差异并不显著；对 *LEV* 且 *NUCS* = 70% (图 41)，*SSGA* 取得最好的统计分析结果，然后是*(1+1)EA*，*AVM* 的统计分析结果最差；对其他情形，*(1+1)EA* 取得最好的统计分析结果，然后是 *SSGA*，而 *AVM* 的统计分析结果最差。

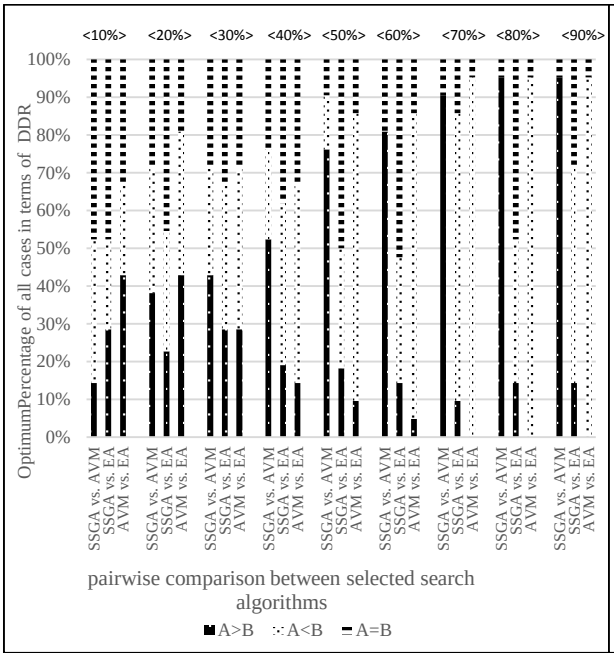

图 36 相似度函数 *CNT* 对应的 *OptimumPercentage*

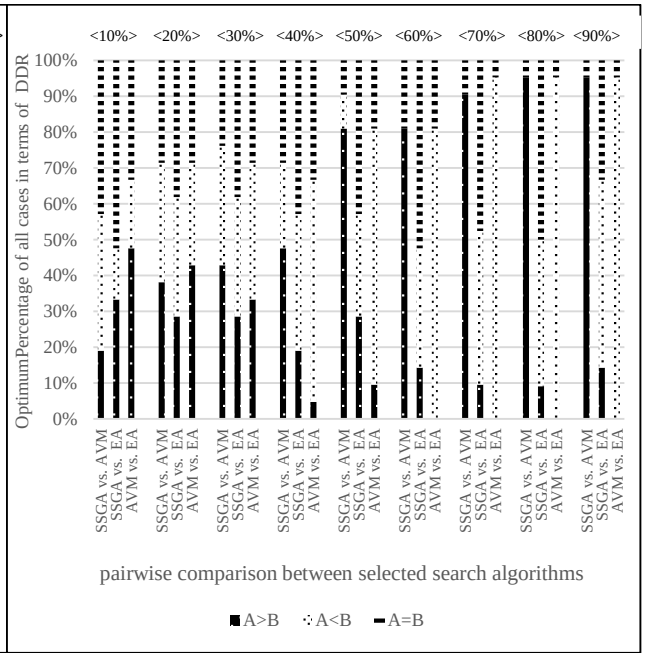

图 38 相似度函数 *GOW* 对应的 *OptimumPercentage*

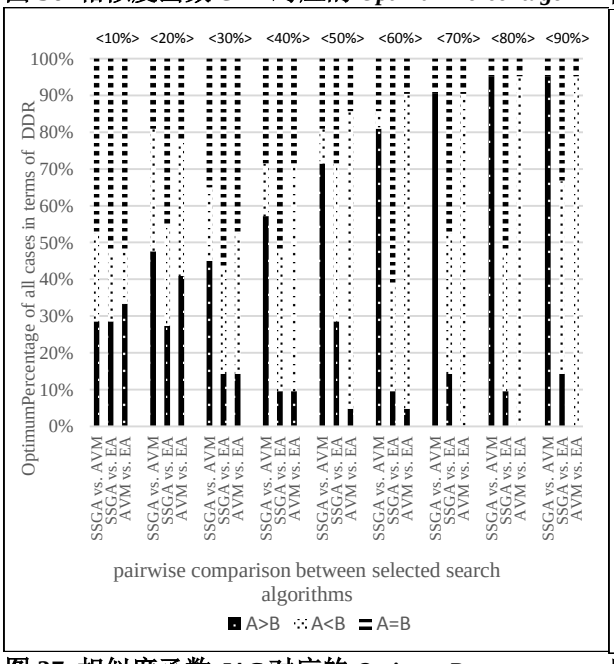

图 37 相似度函数 *JAC* 对应的 *OptimumPercentage*

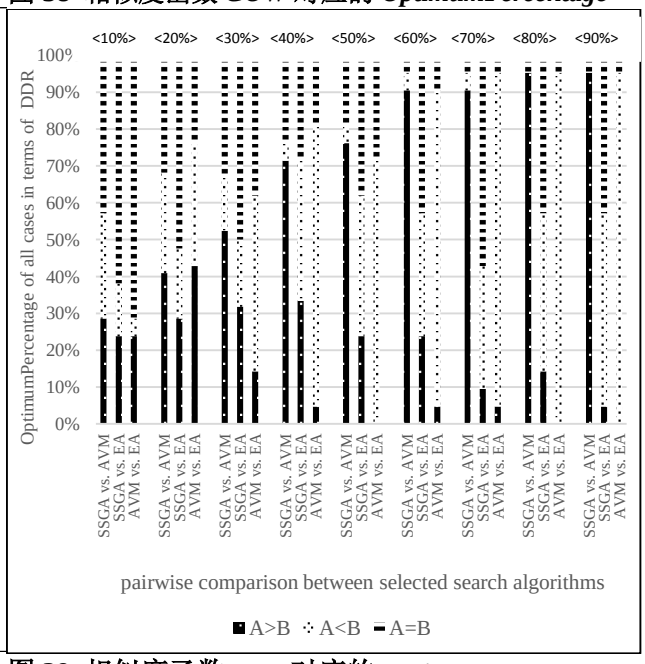

图 39 相似度函数 *SOK* 对应的 *OptimumPercentage*

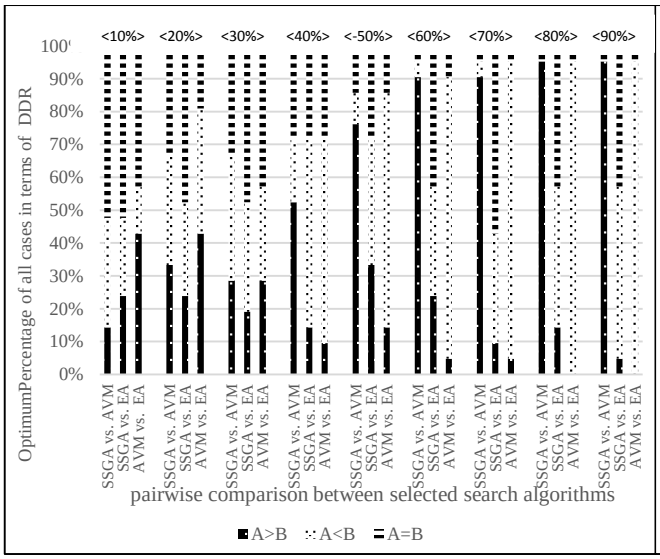

图 40 相似度函数 *NLCS* 对应的 *OptimumPercentage*

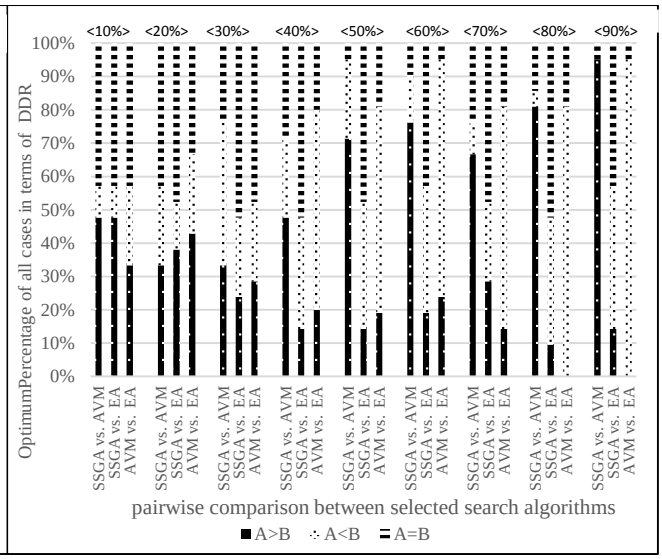

图 41 相似度函数 *LEV* 对应的 *OptimumPercentage*

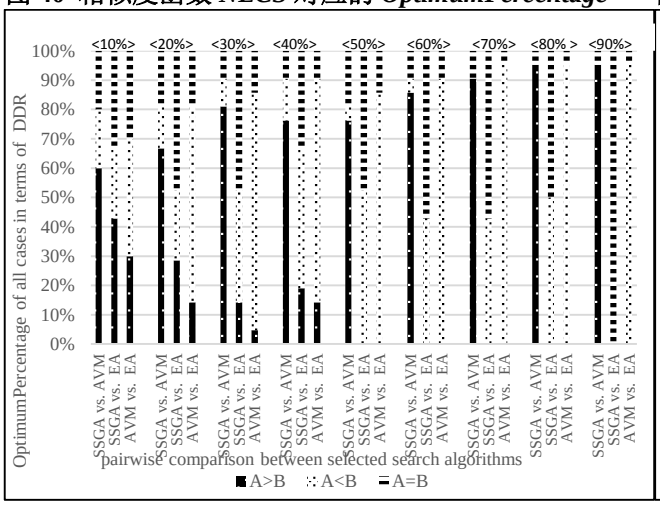

图 42 相似度函数 *NW* 对应的 *OptimumPercentage*

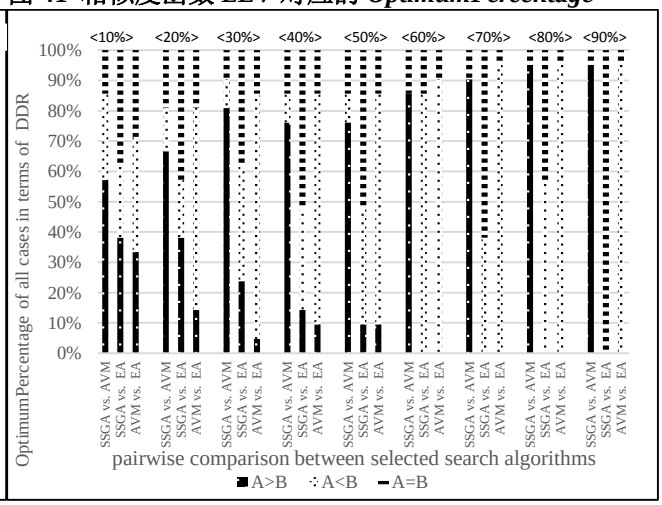

图 43 相似度函数 *SW* 对应的 *OptimumPercentage*

针对 *FV*，RQ2 的详细统计分析结果在附录表 3 中进行了报告，下面对主要的观察发现进行总结。

NUCS = 10%。对相似度函数 *CNT*、*GOW*、*SOK*、*NLCS*、*NW*、*SW*，搜索算法 *SSGA* 取得最好的统计分析结果，*AVM* 和*(1+1)EA* 之间的统计分析差异并不显著。以 *CNT* 为例，如附录表 3 所示，*SSGA vs. AVM*：*SSGA* 能够显著性好地解决 15 个优化问题，而 *AVM* 只能显著性好地解决 2 个优化问题，在剩余的 4 个优化问题，*SSGA* 与 *AVM* 之间并不存在显著性的差异。*SSGA vs. (1+1)EA*：*SSGA* 能够显著性好地解决 16 个优化问题，而*(1+1)EA* 只能显著性好地解决 1 个优化问题，在剩余的 4 个优化问题，*SSGA* 与*(1+1)EA* 之间并不存在显著性的差异。*AVM vs. (1+1)EA*： *AVM* 能够显著性好地解决 8 个优化问题，*(1+1)EA* 能显著性好地解决 8 个优化问题，在剩余的 5 个优化问题，*(1+1)EA* 与 *AVM* 之间并不存在显著性的差异。对相似度函数 *JAC* 和 *LEV*，*SSGA* 取得最好的统计分析结果，然后是*(1+1)EA*，*AVM* 的统计分析结果最差。

NUCS = 20%, 30%。无论使用哪个相似度函数，*SSGA* 取得最好的统计分析结果，然后是*(1+1)EA*，*AVM* 的统计分析结果最差。例如，*JAC* 和 *NUCS* = 20%，*SSGA vs. AVM*：*SSGA* 能够显著性好地解决 20 个优化问题，在剩余的 1 个优化问题 *SSGA* 与 *AVM* 之间并不存在显著性的差异。*SSGA vs. (1+1)EA*：*SSGA* 能够显著性好地解决 17 个优化问题，而*(1+1)EA* 只能显著性好地解决 4 个优化问题。*AVM vs. (1+1)EA*： *AVM* 只能够显著性好地解决 5 个优化问题，*(1+1)EA* 能显著性好地解决 13 个优化问题，在剩余的 3 个优化问题 *(1+1)EA* 与 *AVM* 之间并不存在显著性的差异。针对其他的相似度函数也观察到类似的结果。

NUCS = 40%。无论使用哪个相似度函数，*SSGA* 和*(1+1)EA* 比 *AVM* 要显著得好，而 *SSGA* 和*(1+1)EA* 之间的统计分析差异并不显著。以 *SW* 为例，*SSGA* 能够显著性好地解决 20 个优化问题，在剩余的 1 个优化问题 *SSGA* 与 *AVM* 之间并不存在显著性的差异。相同的结果也在*(1+1)EA* 和 *AVM* 的比较中观察到。*SSGA vs. (1+1)EA*：*SSGA* 能够显著性好地解决 9 个优化问题，*(1+1)EA* 能显著性好地解决 9 个优化问题，在剩余的 3 个优化问题 *SSGA* 与*(1+1)EA* 之间并不存在显著性的差异。针对其他的相似度函数也观察到类似的结果。

NUCS = 50%。对相似度函数 *CNT*、*JAC*、*GOW*、*SOK*，可以发现 *SSGA* 和(1+1) EA 要比 *AVM* 显著得好，而 *SSGA* 和*(1+1)EA* 之间的统计分析差异并不显著。对相似度函数 *NLCS*、*LEV*、*NW*、*SW*， *(1+1)EA* 取得最好的统计分析结果，然后是 *SSGA*，*AVM* 的统计分析结果最差。

NUCS = 60%, 70%, 80%, 90%。无论使用哪个相似度函数，*(1+1)EA* 取得最好的统计分析结果，然后是 *SSGA*，*AVM* 的统计分析结果最差。例如，*CNT* 且 *NUCS* = 60%，*(1+1) EA vs. AVM*： *(1+1)EA* 能够显著性好地解决 20 个优化问题，在剩余的 1 个优化问题 *(1+1)EA* 与 *AVM* 之间并不存在显著性的差异。相同的结果也在 *SSGA* 和 *AVM* 的比较中观察到。*(1+1) EA vs. SSGA*：*(1+1)EA* 能够显著性好地解决 13 个优化问题，*SSGA* 能够显著性好地解决 6 个优化问题，在剩余的 2 个优化问题*(1+1)EA* 与 *SSGA* 之间并不存在显著性的差异。针对其他的相似度函数也观察到类似的结果。

5.4.5.3 RQ3 的实验结果及分析

RQ3 的研究目标是对所有 *CSA* 实例(总共 32 个，即 8 种相似度函数和 4 种搜索算法的组合)的效能进行分析，以确定最好的那一个。为了回答 RQ3，本文以 DDR 为评价指

标，对所有的 *CSA* 实例进行统计分析，即，对每一个CSA_i，将其与其他的 31 个CSA_j以 DDR 为评价指标进行比对分析。首先使用 Kruskal–Wallis 统计方法对所有 *CSA* 实例相应的 DDR 进行检测，如果 Kruskal–Wallis 统计显示存在着显著差异(即p-value 小于 0.05)，则使用 Vargha-Delaney 统计、Wilcoxon 秩和检验同时搭配 Bonferroni 修正方法对每一对CSA_i和CSA_j进行统计比较。其中 Bonferroni 修正方法用于对多重比较中 Wilcoxon 秩和检验返回的 p-value 进行修正，以防止第 I 类统计错误的发生。本节中，针对一个CSA_i，对其比其他CSA_j显著得好的次数进行报告。例如，表 31 中第三行第三列的数值 208 说明$CSA_{CNT-SSGA}$在与其他的CSA_j统计比较中，$CSA_{CNT-SSGA}$有 208 次要显著得好。对每个CSA_i，它总共要进行 31*21 = 651 次统计比较，其中 31 是其他的CSA_j的总数，21 是所有的优化问题的总数。下面对统计分析结果进行总结。

NUCS = 10%。$CSA_{SW-SSGA}$取得最好的统计分析结果。具体地讲，在 651 次的统计比较中，$CSA_{SW-SSGA}$有 538 次要比其他的CSA_j显著得好；有 26 次$CSA_{SW-SSGA}$ 要比其他的CSA_j显著得不好；在剩余的 87 次比较中，$CSA_{SW-SSGA}$与其他的CSA_j之间没有显著性的差异。一个有趣的发现，$CSA_{SW-SSGA}$、$CSA_{SW-(1+1)EA}$、CSA_{SW-AVM}、$CSA_{NW-SSGA}$、$CSA_{NW-(1+1)EA}$、CSA_{NW-AVM}要比其他的CSA_j显著得好，而他们之间却没有显著性的差异。

NUCS = 20%。可以观察到$CSA_{SW-SSGA}$取得了最好的统计分析结果，同时$CSA_{SW-SSGA}$、$CSA_{SW-(1+1)EA}$、$CSA_{NW-SSGA}$、$\mathrm{CSA_{NW-(1+1)EA}}$要比其他的$CSA_j$显著得好。具体地讲，在 651 次的统计比较中，$CSA_{SW-SSGA}$有 543 次要比其他的$CSA_j$显著得好；有 57 次$CSA_{SW-SSGA}$ 要比其他的CSA_j显著得不好；在剩余的 51 次比较中，$CSA_{SW-SSGA}$与其他的CSA_j之间没有显著性的差异。

NUCS = 30%, 40%, 50%, 60%, 70%, 80%。$CSA_{NW-(1+1)EA}$取得了最好的统计分析结果，同时$CSA_{SW-SSGA}$、$CSA_{SW-(1+1)EA}$、$CSA_{NW-SSGA}$、$CSA_{NW-(1+1)EA}$要比其他的CSA_j显著得好。以 *NUCS* = 40%为例，在 651 次的统计比较中，$CSA_{NW-(1+1)EA}$有 558 次要比其他的CSA_j显著得好；有 26 次$CSA_{NW-(1+1)EA}$要比其他的CSA_j显著得不好；在剩余的 67 次比较中，$CSA_{NW-(1+1)EA}$与其他的CSA_j之间没有显著性的差异。

NUCS = 90%。如表 31 所示，$CSA_{SW-SSGA}$、$CSA_{SW-(1+1)EA}$、$CSA_{NW-SSGA}$、$CSA_{NW-(1+1)EA}$比其他的CSA_j显著得好。

表 31 RQ3 统计分析结果(Vargha-Delaney 统计，Wilcoxon 秩和检验和 Bonferroni 修正方法，显著水平 0.05)

NUCS	*SIM*	*SSGA*			*(1+1) EA*			*AVM*			*RS*		
		A>B	**A<B**	**A=B**	**A>B**	**A<B**	**A=B**	**A>B**	**A<B**	**A=B**	**A>B**	**A<B**	**A=B**
10%	***CNT***	208	254	189	192	298	161	236	233	182	122	344	185
	JAC	242	231	178	212	241	198	235	223	193	144	321	186
	GOW	208	251	192	181	283	187	234	236	181	133	330	188
	SOK	227	241	183	209	242	200	234	221	196	166	306	179
	NLCS	174	294	183	181	309	161	191	263	197	135	347	169
	LEV	90	454	107	48	510	93	41	512	98	38	519	94
	NW	**535**	31	85	**524**	53	74	**530**	64	57	394	159	98
	SW	**538**	26	87	**524**	53	74	**528**	60	63	407	151	93
20%	***CNT***	260	258	133	272	246	133	292	247	112	137	375	139
	JAC	278	273	100	280	276	95	284	226	141	122	371	158
	GOW	259	258	134	264	250	137	290	231	130	133	374	144
	SOK	306	245	100	272	277	102	297	223	131	140	355	156
	NLCS	229	275	147	230	277	144	257	268	126	108	395	148
	LEV	53	533	65	35	558	58	52	520	79	20	566	65
	NW	**539**	59	53	**534**	56	61	499	88	64	339	231	81
	SW	**543**	57	51	**537**	51	63	496	92	63	343	218	90
30%	***CNT***	261	266	124	273	260	118	236	264	151	98	393	160
	JAC	333	201	117	337	199	115	268	213	170	94	378	179
	GOW	255	264	132	251	261	139	242	255	154	91	396	164
	SOK	387	158	106	374	171	106	271	219	161	111	367	173
	NLCS	218	321	112	224	297	130	216	273	162	75	422	154
	LEV	74	518	59	86	489	76	71	497	83	18	575	58
	NW	**554**	49	48	**570**	19	62	483	118	50	266	280	105
	SW	**558**	46	47	**564**	23	64	467	116	68	265	283	103
40%	***CNT***	271	250	130	297	216	138	220	235	196	67	397	187
	JAC	369	186	96	376	176	99	209	239	203	70	423	158
	GOW	266	264	121	302	207	142	213	248	190	72	399	180
	SOK	446	140	65	437	153	61	211	250	190	65	403	183
	NLCS	278	259	114	325	206	120	194	273	184	61	420	170
	LEV	193	396	62	171	423	57	113	420	118	12	535	104
	NW	**538**	49	64	**558**	26	67	346	190	115	137	370	144
	SW	**537**	48	66	**551**	26	74	348	192	111	141	375	135
50%	***CNT***	352	212	87	372	188	91	194	288	169	51	441	159
	JAC	391	179	81	440	149	62	183	284	184	44	438	169
	GOW	350	201	100	355	199	97	175	297	179	44	439	168
	SOK	411	151	89	431	149	71	189	285	177	53	425	173
	NLCS	352	219	80	361	209	81	193	301	157	39	464	148
	LEV	268	332	51	266	342	43	144	392	115	19	515	117
	NW	**531**	46	74	**586**	0	65	272	239	140	97	402	152
	SW	**521**	57	73	**536**	40	75	276	237	138	102	391	158
60%	***CNT***	372	154	125	416	141	94	160	301	190	44	441	166
	JAC	419	140	92	464	124	63	169	287	195	35	450	166
	GOW	369	162	120	412	128	111	168	306	177	36	447	168
	SOK	450	138	63	463	132	56	165	297	189	41	440	170
	NLCS	362	178	111	394	172	85	165	322	164	35	451	165
	LEV	285	336	30	304	319	28	136	378	137	8	517	126
	NW	**515**	76	60	**592**	0	59	192	291	168	53	429	169
	SW	**514**	73	64	**559**	0	92	196	299	156	46	432	173
70%	***CNT***	391	133	127	406	141	104	197	293	161	35	434	182
	JAC	418	122	111	433	133	85	164	298	189	33	443	175
	GOW	375	153	123	409	139	103	184	300	167	34	442	175
	SOK	426	134	91	421	167	63	185	295	171	36	431	184
	NLCS	357	175	119	377	198	76	173	302	176	33	445	173
	LEV	295	277	79	305	305	41	172	301	178	29	448	174
	NW	**494**	78	79	**573**	0	78	186	299	166	29	444	178
	SW	**490**	78	83	**568**	0	83	194	294	163	32	445	174
80%	***CNT***	400	109	142	452	68	131	151	332	168	3	471	177
	JAC	412	98	141	477	56	118	134	339	178	10	475	166
	GOW	403	108	140	453	64	134	139	337	175	3	479	169
	SOK	420	93	138	481	45	125	157	332	162	2	480	169
	NLCS	397	121	133	445	83	123	145	331	175	1	489	161
	LEV	323	266	62	350	244	57	167	331	153	2	490	159
	NW	443	76	132	**532**	0	119	138	335	178	6	477	168
	SW	443	86	122	**519**	0	132	141	334	176	10	477	164
90%	***CNT***	381	107	163	447	34	170	151	340	160	1	488	162
	JAC	381	102	168	456	41	154	151	340	160	4	497	150
	GOW	378	100	173	452	38	161	147	336	168	1	486	164
	SOK	374	95	182	460	21	170	158	336	157	6	476	169
	NLCS	382	103	166	450	44	157	152	342	157	5	491	155
	LEV	353	170	128	405	148	98	156	339	156	2	490	159
	NW	**493**	0	158	**495**	0	156	149	342	160	5	476	170
	SW	**478**	0	173	**489**	0	162	140	336	175	3	485	163

基于以上统计分析结果，可以发现$CSA_{SW-SSGA}$、$CSA_{SW-(1+1)EA}$，、$CSA_{NW-SSGA}$，、

$CSA_{NW-(1+1)EA}$要比其他的CSA_j显著得好。具体地讲，当 *NUCS*=10%，$CSA_{SW-(1+1)EA}$取得最好的统计结果；当 *NUCS*=20%，$CSA_{SW-SSGA}$取得最好的统计结果；当 *NUCS* = 30%，40%，50%... 80%，$CSA_{NW-(1+1)EA}$取得最好的统计结果；当 *NUCS* = 90%， $CSA_{SW-SSGA}$、$CSA_{SW-(1+1)EA}$、$CSA_{NW-SSGA}$、$CSA_{NW-(1+1)EA}$取得最好的统计结果。

5.4.5.4 RQ4 的实验结果及分析

为了回答 RQ4，采用 Kendall's Tau 统计检验对 *FV* 和 *DDR* 之间的相关性进行检验。表 32 报告了详细的统计结果，下面进行总结。

针对相似度函数 *CNT*，在 *FV* 和 *DDR* 之间存着显著的负相关性。以 *NUCS* = 10%为例，$\tau = -0.85$且p-value < 0.05，这说明在 *FV* 和 *DDR* 之间存着显著的负相关性，同时暗含说明了，给定数量 *m* 下，一组 RUCM 用况场景如果其相似度越低则其缺陷检测能力越强。类似的结果在 *NUCS* 的其他取值(即 20%, 30%, 40%...90%)通用可以观察到。对其他的相似度函数(即 JAC、GOW、SOK、NLCS、LEV、NW、SW)，可以观察到与 *CNT* 类似的结果。

基于表 32 中的统计分析结果，可以得出以下结论：在 *FV* 和 *DDR* 之间存着显著的负相关性，无论哪个相似度函数被使用还是 *NUCS* 取何值(10%, 20%, 30%, 40%...90%)，因为所有的τ值都小于-0.5 同时相应的p-values 都小于 0.05。

表 32 *FV* 和 *DDR* 相关性检验(Kendall's Tau 检验，显著水平 0.05)

SIM		***NUCS***								
		10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
CNT	τ	-0.85	-0.86	-0.71	-0.93	-0.93	-0.69	-0.79	-0.502	-0.54
	p-value	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
JAC	τ	-1	-0.86	-0.79	-0.93	-0.86	-0.85	-0.79	-0.53	-0.55
	p-value	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
GOW	τ	-0.86	-0.79	-0.64	-0.93	-0.93	-0.70	-0.79	-0.51	-0.55
	p-value	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
SOK	τ	-0.91	-1	-0.71	-0.93	-0.93	-0.82	-0.93	-0.52	-0.55
	p-value	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
NLCS	τ	-0.71	-0.71	-0.57	-1	-0.79	-0.79	-0.86	-0.51	-0.58
	p-value	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
LEV	τ	-0.57	-0.67	-0.58	-0.5	-0.64	-0.64	-0.58	-0.46	-0.56
	p-value	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
NW	τ	-0.64	-0.57	-0.57	-0.86	-0.79	-0.79	-0.79	-0.57	-0.62
	p-value	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
SW	τ	-0.64	-0.57	-0.5	-0.86	-0.79	-0.76	-0.86	-0.56	-0.61
	p-value	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05

5.4.5.5 RQ5 的实验结果及分析

该研究问题被设计来对 S^{3}RUCM 的实际使用进行评估，即可以期望选择多少个

RUCM 用况场景以达到最大的需求缺陷发现率。为了进行全面的分析研究，本文针对每一个CSA_i，采用 *EFF*(见 5.4.3.3 节)为评价指标对其进行分析，同时也针对 *NUCS* 的每个取值给了CSA_i相应的 DDR。详细的实验结果在表 33 中进行报告，下面对其进行分析总结。

表 33 CSA_i的功效分析结果*

CSA		*EFF*									*DDR*								
		10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
CNT	***SSGA***	0.53	0.51	0.46	0.42	0.4	0.39	0.35	0.34	0.25	0.18	0.35	0.47	0.57	0.68	0.8	0.85	0.93	0.94
	EA	0.51	0.52	0.57	0.43	0.41	0.4	0.36	0.35	0.31	0.18	0.35	0.58	0.58	0.69	0.81	0.86	0.95	0.96
	AVM	0.54	0.51	0.44	0.39	0.34	0.32	0.29	0.27	0.26	0.18	0.35	0.45	0.53	0.58	0.66	0.68	0.75	0.79
	RS	0.49	0.46	0.4	0.35	0.31	0.3	0.26	0.25	0.24	0.17	0.31	0.41	0.48	0.53	0.61	0.63	0.69	0.74
JAC	***SSGA***	0.55	0.51	0.47	0.44	0.41	0.4	0.36	0.34	0.31	0.19	0.35	0.48	0.6	0.69	0.82	0.86	0.94	0.94
	EA	0.53	0.51	0.47	0.44	0.42	0.4	0.36	0.35	0.31	0.18	0.35	0.48	0.6	0.71	0.83	0.86	0.96	0.96
	AVM	0.55	0.51	0.45	0.39	0.34	0.32	0.28	0.27	0.26	0.19	0.35	0.46	0.53	0.58	0.66	0.68	0.74	0.79
	RS	0.5	0.46	0.4	0.35	0.31	0.3	0.26	0.25	0.24	0.17	0.31	0.41	0.48	0.53	0.61	0.63	0.69	0.74
GOW	***SSGA***	0.53	0.51	0.45	0.42	0.4	0.39	0.35	0.34	0.31	0.18	0.35	0.46	0.57	0.68	0.8	0.84	0.94	0.94
	EA	0.51	0.52	0.46	0.43	0.4	0.4	0.36	0.35	0.31	0.18	0.35	0.47	0.59	0.69	0.81	0.85	0.95	0.96
	AVM	0.54	0.51	0.44	0.39	0.34	0.32	0.29	0.27	0.26	0.18	0.35	0.45	0.53	0.58	0.66	0.68	0.75	0.79
	RS	0.49	0.46	0.4	0.35	0.31	0.3	0.26	0.25	0.24	0.17	0.31	0.41	0.48	0.53	0.61	0.62	0.69	0.74
SOK	***SSGA***	0.54	0.52	0.48	0.45	0.41	0.4	0.36	0.35	0.31	0.18	0.35	0.49	0.61	0.71	0.83	0.85	0.94	0.94
	EA	0.52	0.51	0.48	0.45	0.42	0.41	0.36	0.35	0.31	0.18	0.35	0.49	0.62	0.72	0.83	0.86	0.96	0.96
	AVM	0.54	0.52	0.45	0.39	0.34	0.32	0.29	0.27	0.26	0.18	0.35	0.46	0.53	0.58	0.66	0.68	0.75	0.79
	RS	0.5	0.46	0.4	0.35	0.32	0.3	0.26	0.25	0.24	0.17	0.31	0.41	0.48	0.54	0.61	0.62	0.69	0.74
NLCS	***SSGA***	0.51	0.5	0.43	0.38	0.39	0.38	0.35	0.34	0.31	0.17	0.34	0.44	0.51	0.67	0.79	0.84	0.93	0.94
	EA	0.5	0.5	0.45	0.42	0.4	0.39	0.35	0.35	0.31	0.17	0.34	0.46	0.57	0.68	0.79	0.84	0.94	0.96
	AVM	0.52	0.49	0.44	0.41	0.34	0.32	0.28	0.28	0.26	0.18	0.34	0.45	0.55	0.57	0.65	0.68	0.75	0.79
	RS	0.49	0.44	0.39	0.34	0.31	0.3	0.26	0.25	0.24	0.17	0.3	0.4	0.47	0.53	0.61	0.62	0.69	0.73
LEV	***SSGA***	0.41	0.4	0.36	0.36	0.34	0.34	0.32	0.32	0.3	0.14	0.27	0.37	0.48	0.59	0.7	0.76	0.87	0.93
	EA	0.39	0.38	0.36	0.36	0.35	0.35	0.32	0.32	0.31	0.13	0.26	0.37	0.48	0.59	0.71	0.76	0.89	0.94
	AVM	0.4	0.39	0.37	0.34	0.32	0.31	0.28	0.28	0.26	0.14	0.27	0.37	0.46	0.54	0.62	0.68	0.75	0.79
	RS	0.38	0.37	0.34	0.31	0.29	0.28	0.26	0.25	0.24	0.13	0.25	0.34	0.42	0.5	0.58	0.63	0.69	0.74
NW	***SSGA***	0.85	**0.82**	0.8	0.7	0.57	0.48	0.41	**0.37**	**0.33**	**0.29**	**0.56**	0.81	0.96	0.96	0.98	0.99	**1**	**1**
	EA	0.84	0.81	**0.83**	**0.72**	**0.59**	**0.49**	**0.42**	**0.37**	**0.33**	**0.29**	0.55	**0.85**	**0.99**	**1**	**1**	**1**	**1**	**1**
	AVM	0.83	0.71	0.56	0.43	0.36	0.32	0.29	0.27	0.26	0.28	0.48	0.57	0.59	0.61	0.66	0.68	0.74	0.78
	RS	0.68	0.58	0.46	0.38	0.32	0.3	0.26	0.25	0.24	0.23	0.39	0.47	0.52	0.55	0.62	0.63	0.69	0.74
SW	***SSGA***	**0.86**	**0.82**	0.77	0.69	0.57	0.48	0.41	**0.37**	**0.33**	**0.29**	**0.56**	0.78	0.94	0.96	0.98	0.99	**1**	**1**
	EA	0.84	0.81	0.81	0.71	0.58	**0.49**	**0.42**	**0.37**	**0.33**	**0.29**	0.55	0.83	0.97	0.99	**1**	**1**	**1**	**1**
	AVM	0.84	0.71	0.56	0.43	0.36	0.32	0.29	0.27	0.26	**0.29**	0.48	0.57	0.59	0.61	0.66	0.68	0.75	0.79
	RS	0.69	0.58	0.46	0.38	0.32	0.3	0.26	0.25	0.24	0.23	0.4	0.47	0.52	0.55	0.62	0.63	0.69	0.74

*EA: (1+1) EA

NUCS = 10%。*EFF* 的结果值在 0.38 到 0.86 之间变动，而 *DDR* 的结果值在 0.13 到 0.29 之间变动。在所有的 32 个 *CSA* 实例中，$CSA_{SW-SSGA}$取得了最好的 *EFF* 值(即 0.86)，暗含说明了，$CSA_{SW-SSGA}$返回的 RUCM 用况场集合中，每个 RUCM 用况场景平均可以检

测出 0.86 个需求缺陷。所有的相似度函数中，*NW* 和 *SW* 表现的最好，如表 33 所示，它们的 *EFF* 值从 0.68 到 0.86，而其他的相似度函数的 *EFF* 值从 0.41 到 0.55。

NUCS = 20%。*EFF* 的结果值在 0.37 到 0.82 之间变动，而 *DDR* 的结果值在 0.25 到 0.56 之间变动。$CSA_{SW-SSGA}$和$CSA_{NW-SSGA}$表现的最好。相似度函数 *LEV* 的性能表现最差，它的 *EFF* 值从 0.37 到 0.4，*DDR* 取值从 0.25 到 0.27。所有的相似度函数中，*NW* 和 *SW* 表现的最好，它们的 *EFF* 值从 0.58 到 0.82，*DDR* 取值从 0.39 到 0.56，而其他的相似度函数的 *EFF* 值从 0.44 到 0.52，*DDR* 取值从 0.3 到 0.35。

NUCS = 30%, 40%, 50%。$CSA_{NW-(1+1)EA}$的表现最好，无论是以 *EFF* 进行评价还是以 *DDR* 进行评价。当 *NUCS* 取值达到 50%，$CSA_{NW-(1+1)EA}$取得了 *DDR* 的最大值 100%，同时其 *EFF* 值为 0.59，这暗示说明$CSA_{NW-(1+1)EA}$通过选取 50%的全部 RUCM 用况场景可以检测出所有注入的 RUCM 需求缺陷，而每个 RUCM 用况场景评价可以发现 0.59 个需求缺陷。

NUCS = 60%, 70%。$CSA_{NW-(1+1)EA}$和$CSA_{SW-(1+1)EA}$表现的最好，*DDR* 取得了最大值 100%。

NUCS = 80%, 90%。$CSA_{NW-(1+1)EA}$、$CSA_{NW-SSGA}$、$CSA_{SW-(1+1)EA}$、$CSA_{SW-SSGA}$表现的最好， *DDR* 取得了最大值 100%。

基于以上统计结果可以得出以下结论：给定一个 *NUCS*，*SW* 和 *NW* 要比其他的相似度函数表现的显著得好。在所有那些能够取得 *DDR*=100%的 *CSA* 实例中，$CSA_{NW-(1+1)EA}$是最好的，因为它只需要选择 50%的全部 RUCM 用况场景便能取得最好的 *EFF* 数据值 0.59。

5.4.5.6 RQ6 的实验结果及分析

本小节对每个CSA_i的平均执行时间进行分析，其中详细的数据在附录表 4 进行报告。基于附录表 4 创建图 44-图 47，其中图 44 描绘了由*(1+1)EA* 与各个相似度函数(即 *CNT*、*JAC*、*GOW*、*SOK*、*NLCS*、*LEV*、*NW*、*SW*)构成的各个CSA_i的平均执行时间。图 45 描绘了由 *RS* 与各个相似度函数(即 *CNT*、*JAC*、*GOW*、*SOK*、*NLCS*、*LEV*、*NW*、*SW*)构成的各个CSA_i的平均执行时间。图 46 描绘了由 *AVM* 与各个相似度函数(即 *CNT*、*JAC*、*GOW*、*SOK*、*NLCS*、*LEV*、*NW*、*SW*)构成的各个CSA_i的平均执行时间。图 47 描绘了由 *SSGA* 与各个相似度函数(即 *CNT*、*JAC*、*GOW*、*SOK*、*NLCS*、*LEV*、*NW*、*SW*)构成的各个CSA_i的平均执行时间。

对于每一个CSA_i，其相应的 *RT*(即，平均执行时间) 数值随着 *NUCS* 取值的增大而增加。以$CSA_{CNT-(1+1)EA}$为例，如图 44 所示，当 *NUCS* = 10%，$CSA_{CNT-(1+1)EA}$平均耗时 3 秒，而当 *NUCS* = 90%，$CSA_{CNT-(1+1)EA}$平均耗时达到 271 秒。同样的观察结果也适应于其他的CSA_i。

对于每一个相似度函数，给定一个 *NUCS*，各个搜索算法表现出不同的时间性能。以 *CNT* 且 *NUCS* = 20%为例，如图 44 和图 45 所示，*(1+1)EA* 的平均耗时是 13 秒，而 *RS* 的平均耗时是 14 秒。同样的观察结果也适应于其他的相似度函数。

对于每一个搜索算法，给定一个 *NUCS*，基于集合的相似度计算函数(即，*CNT*、*JAC*、*GOW*、*SOK*)的时间消耗要比基于序列的相似度计算函数(即，*NLCS*、*LEV*、*NW*、*SW*)的时间消耗小。例如，图 44 所示，当*(1+1)EA* 且 *NUCS* = 30%()，*CNT* 的平均耗时是 27 秒而 *SW* 的平均耗时是 65 秒。同样的观察结果也适应于其他的搜索算法。

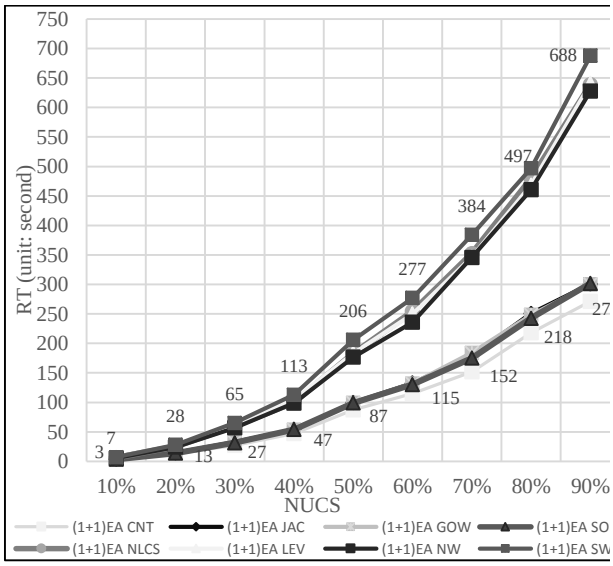

图 44 *CSA* 实例的平均执行时间(*(1+1)EA*)

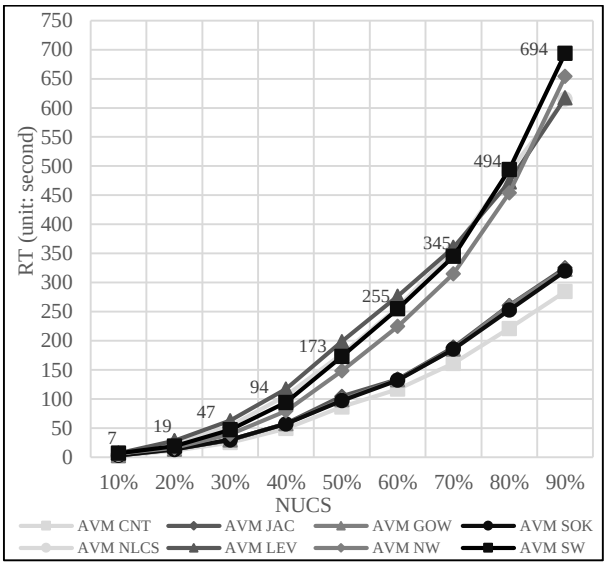

图 46 *CSA* 实例的平均执行时间(*AVM*)

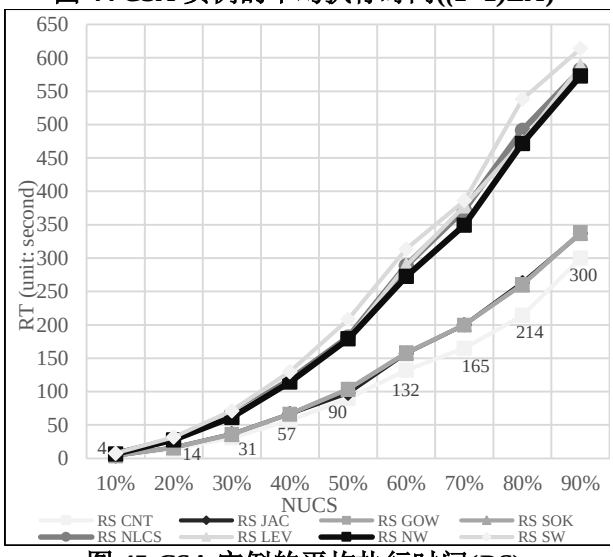

图 45 *CSA* 实例的平均执行时间(*RS*)

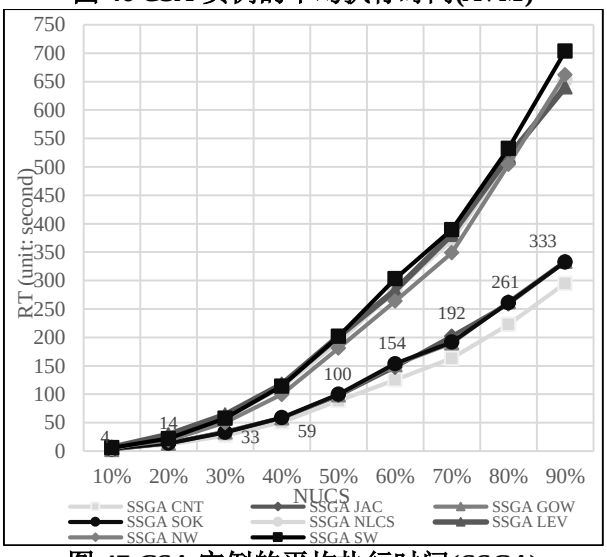

图 47 *CSA* 实例的平均执行时间(*SSGA*)

5.4.6 实验讨论

5.4.6.1 RQ1 和 RQ2 的相关讨论

基于 RQ1 的实验数据，本文所要解决的 RUCM 用况场景选择问题并不是一个朴素问题，因此需要寻求一种有效的方法。同时可以观察到，给定一个相似度计算函数，S^3RUCM 中采用的搜索算法 *SSGA*、*(1+1)EA*、AVM 都要比 RS 显著得好，无论是以 *FV* 进行评估还是以 *DDR* 进行评估。因此可以得出结论，采用基于搜索的方法是解决 RUCM 用况场景选择的一种有效方法。

采用 *DDR* 评价指标对每一个搜索算法进行剖析时(即，RQ2)，可以观察到相似度函数和 *NUCS* 的取值都会影响其效能。例如，当相似度函数采用 *CNT*、*JAC*、*SOK*、*NLCS* 而 *NUCS* = 10%，在绝大多数的优化问题中，各个搜索算法的效能统计分析并没有显著差异。此时最有可能的解释是：当 *NUCS*=10%，构成候选方案的 RUCM 用况场景相比较全部生产的 RUCM 用况场景非常小(仅占 10%)，因此局部搜索 *AVM* 与全局搜索 *SSGA*、*(1+1)EA* 之间的统计比较并不显著。然而当 *NUCS* > 30%，无论 *SSGA* 还是*(1+1)EA* 它们与 *AVM* 的统计比较都要显著得好。更进一步，*NUCS* = 40%、50%、60%、70%、80%、90%时，*(1+1)EA* 取得最好的统计分析结果，然后是 *SSGA*，*AVM* 的统计分析结果最差。这是因为，与 *AVM* 相比，(1+1)EA 是一个全局搜索的方法因此可以找到更好的优化方案；尽管 *SSGA* 也是一个全局搜索方法，它的搜索过程同时依赖于交叉操作和变异操作，因此当 *NUCS* = 40%、50%、60%、70%、80%、90%时，SSGA 可能需要借助交叉操作产生更多的子代来获取一个优化方案。而*(1+1)EA* 仅需要进行变异操作，因此可能会更快速的发现一个优化方案。

当以 *FV* 为评价指标对各个相似度函数进行分析时，可以获得类似的结果。具体地讲，当 *NUCS* = 10%、20%、30%时，*SSGA* 取得最好的统计分析结果；*NUCS* = 40%，*SSGA* 和*(1+1)EA* 之间的统计差异并不显著；当 *NUCS* = 550%、60%、70%、80%、90%时，在绝大多数的优化问题中，*(1+1)EA* 取得最好的统计分析结果，而 *AVM* 从未取得过最好的统计分析结果，这暗示了全局搜索算法要比局部搜索算法更适应于解决 RUCM 用况场景的选择问题。

5.4.6.2 RQ3 的相关讨论

图 48 依据CSA_i的所有 *DDR* 数据为其创建相应的箱线图。给定一个CSA_i，针对 *NUCS*

的每一个取值，将CSA_i对每一个优化问题的 *DDR* 数据进行收集，最后创建图 48 的箱线图对其进行分析。箱线图中的“上须”表示取得的最高 *DDR* 值，“下须”表示取得最低 *DDR* 值，“中线”表示所有 *DDR* 取值的中位数，灰色箱盒描述了 50%的 *DDR* 取值的分布。

如图 48 所示，可以发现$CSA_{NW-(1+1)EA}$(标记为‘7E’)取得最好的表现，然后是$CSA_{SW-(1+1)EA}$(标记为‘8E’)。具体地讲，$CSA_{NW-(1+1)EA}$取得最高的 *DDR* 值，即 100%，同时 50%的 *DDR* 返回值 分布在 85%-100%。有两个离群点(标记为‘。’)，它们分别是 *NUCS* = 10%时的 DDR 值和 *NUCS* = 20%是的 DDR 值。当对$CSA_{NW-(1+1)EA}$与其他CSA_i进行对比时，只有$CSA_{SW-(1+1)EA}$也取得了最好的 *DDR* 值，然而$CSA_{SW-(1+1)EA}$的 *DDR* 返回值中有 50%都分布在 81%-100%，这要比$CSA_{NW-(1+1)EA}$差。而剩余其他的CSA_i，它们的 *DDR* 返回值的 50%仅仅分布在 34%-86%。图 48 显示的结论与 5.4.5.3 节中 RQ3 的实验结果一致。基于以上数据分析，本文推荐采用$CSA_{NW-(1+1)EA}$进行工业实践。

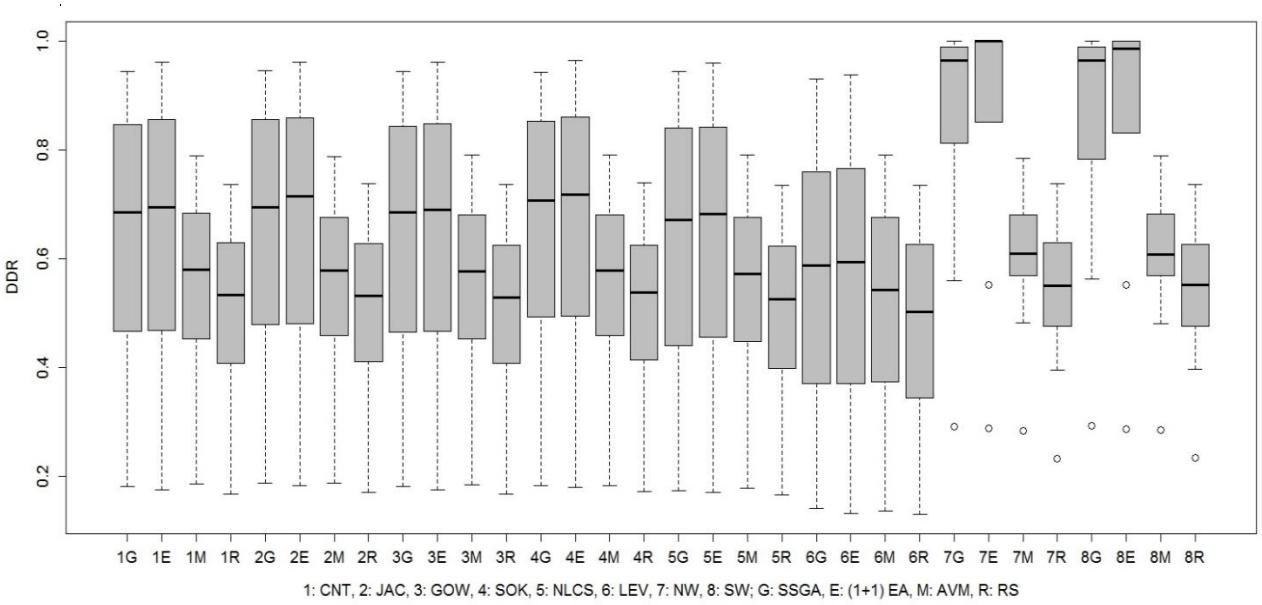

图 48 32 个 CSA 实例的箱线图(以 *DDR* 为评价指标)

5.4.6.3 RQ4 和 RQ5 的相关讨论

RQ4 主要对 *FV* 和 *DDR* 之间的相关性进行判定。实验数据的统计分析显示在 *FV* 和 *DDR* 之间存在显著的负相关性。给定一个 *NUCS* 取值，平均相似度越小的一组 RUCM 用况场景能够发现更多的需求缺陷。

依据 5.4.5.5 节的实验数据，当 *NUCS* = 50%时，$CSA_{NW-(1+1)EA}$检测出了所有注入的需求缺陷。图 49-图 52，针对基于集合的相似度函数，给出了 *DDR* 随着 *NUCS* 不同取值的动态变化；图 53-图 56，针对基于序列的相似度函数，给出了 *DDR* 随着 *NUCS* 不

同取值的动态变化。

对相似度函数 *CNT*，如图 49 所示，在大多数的优化问题中，*SSGA* 和*(1+1)EA* 的表现要显著得好，而 RS 的表现最差。当 NUCS 超过 20%时，*AVM*、*SSGA*、*(1+1)EA* 之间的统计分析差异变得显著。随着 *NUCS* 取值的增大，*DDR* 的取值也在增加，但是 DDR 从未取到 1 而是在 0.95 结束。这可能是因为 *CNT* 是一种基于集合的相似度计算函数，它并不考虑 RUCM 用况场景中 RUCM 需求描述语句的出现次序和重复出现的情形。对于其他的基于集合的相似度函数(图 50-图 52)，可以得到相似的结论。

图 53 针对相似度函数 NLCS，描述了 *DDR* 随着 *NUCS* 不同取值的动态变化，可以得出与前面基于集合的相似度函数的类似结论。对相似度函数 *LEV*，如图 54 所示，DDR 取值的变化趋势与基于集合的相似度函数的变化类似。图 55 针对相似度函数 *NW*，描述了 *DDR* 随着 *NUCS* 不同取值的动态变化。可以发现，所有选用的搜索算法(*SSGA*、*(1+1)EA*、*AVM*)表现的都要比 *RS* 显著得好，无论 *NUCS* 取何值；*(1+1)EA* 表现的最好，然后是 *SSGA*，*AVM* 最差。具体讲，*(1+1)EA* 通过选择 50%的全部 RUCM 用况场景发现所有注入的需求缺陷(即 *DDR*=1)，而 SSGA 需要选择 80%的全部 RUCM 用况场景才可以取得 *DDR*=1。*AVM* 和 *RS* 从未达到过 *DDR*=1。同样的趋势可以在相似度函数 *SW*(图 56)的分析中发现，其中(1+1)EA 通过选择 60%的全部 RUCM 用况场景发现所有注入的需求缺陷(即 *DDR*=1)。

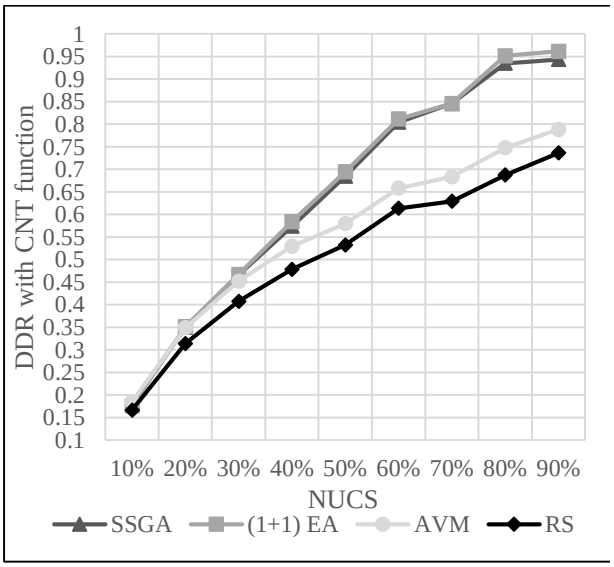

图 49 相似度函数 *CNT* 对应的 *DDR*

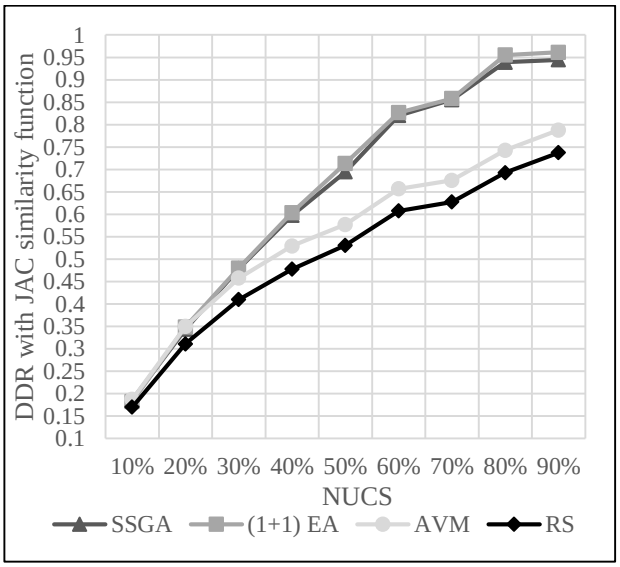

图 50 相似度函数 *JAC* 对应的 *DDR*

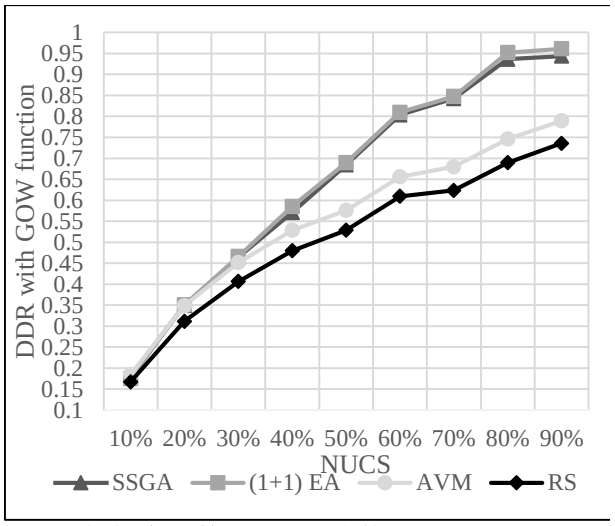

图 51 相似度函数 *GOW* 对应的 *DDR*

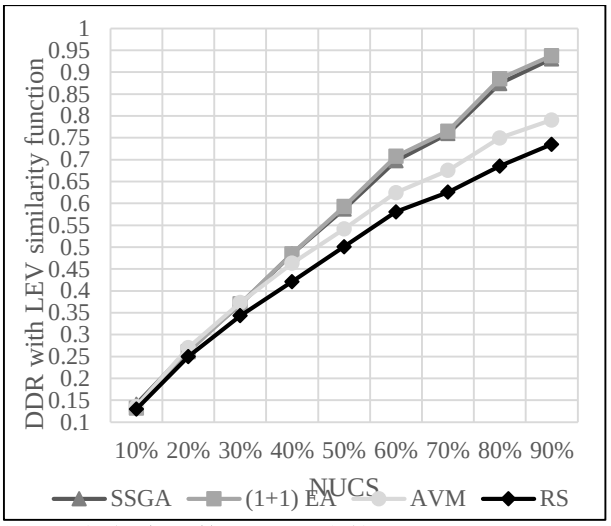

图 54 相似度函数 *LEV* 对应的 *DDR*

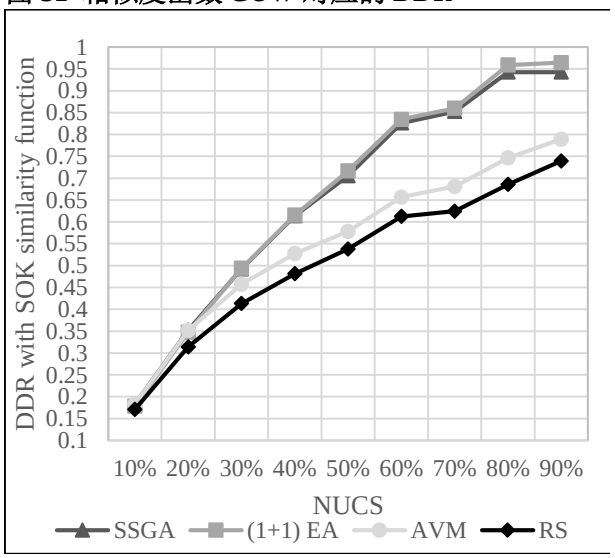

图 52 相似度函数 *SOK* 对应的 *DDR*

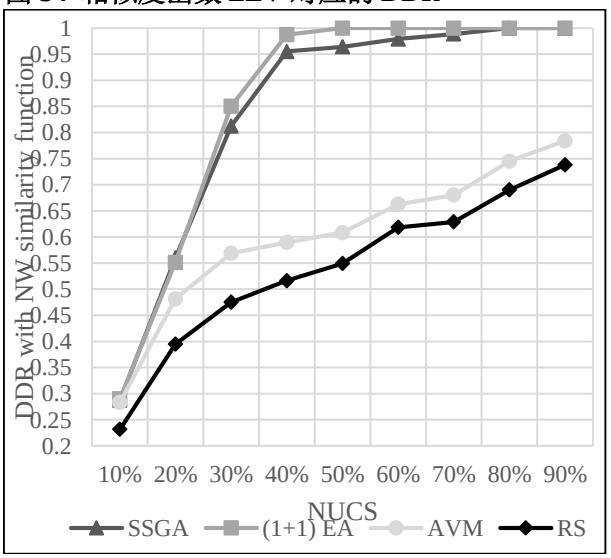

图 55 相似度函数 *NW* 对应的 *DDR*

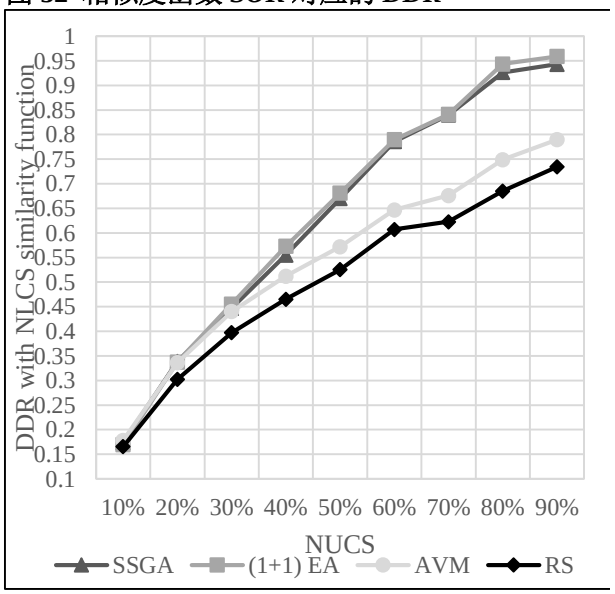

图 53 相似度函数 *NLCS* 对应的 *DDR*

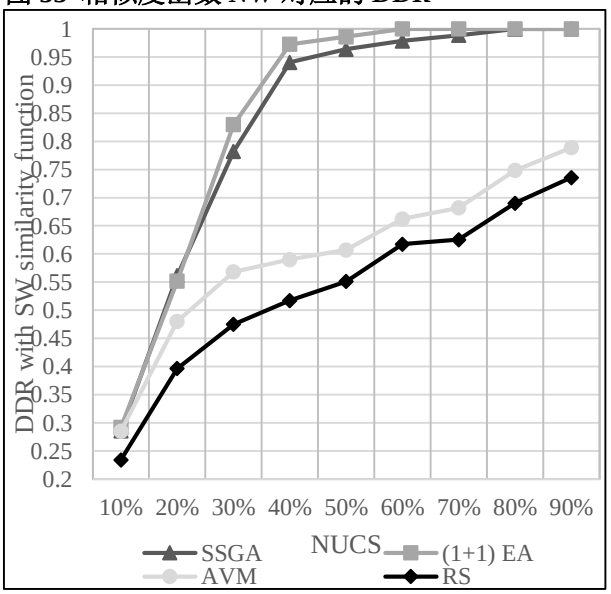

图 56 相似度函数 *SW* 对应的 *DDR*

在所有的 32 个 CSA 实例中，如图 49 -图 56 所示，可以发现$CSA_{NW-(1+1)EA}$取得最好的功效，即它可以选择 50%的全部 RUCM 用况场景发现所有注入的需求缺陷(即 *DDR*=1)。 *(1+1)EA* 是一个全局搜索算法，它可以针对搜索空间给出全局最好的优化方案； *NW* 是一个基于序列的相似度计算函数，它不仅考虑元素的出现次序同时也考虑元素的重现，因此与基于集合的相似度函数相比(如，*CNT*、*JAC*)，*NW* 更适合于 RUCM 用况场景的选择。当对基于序列的相似度函数进行分析时，*NW* 和 *SW* 不仅要考虑元素的匹配、元素的不匹配，同时还要考虑元素的空缺操作，因此它们相比较 *NLCS* 和 *LEV* 而言要更加精确。此外，*NW* 是一个全局序列比对算法，而 *SW* 是一个局部序列比对方法，因此 *NW* 能够取得更好的表现。既然*(1+1)EA* 是最好的全局搜索算法而 *NW* 是最好的全局序列比对算法，那么$CSA_{NW-(1+1)EA}$可以取得最好的功效。因此本文推荐$CSA_{NW-(1+1)EA}$且 *NUCS* = 50% 进行工业实践。

5.4.6.4 RQ6 的相关讨论

针对每一个CSA_i，对其平均执行时间 *RT* 进行评估。如图 44-图 47 所示，可以发现基于集合的相似度函数(*CNT*、*JAC*、*GOW*、*SOK*)要比基于序列的相似度函数(*NLCS*、*LEV*、*NW*、*SW*)耗时少。基于序列的相似度函数都是动态规划算法，它们需要更多的时间对一对 RUCM 用况场景的相似度进行计算，因为基于序列的相似度函数需要考虑序列元素的次序和重复出现。相应地，基于序列的相似度函数返回的 RUCM 用况场景集合能够发现更多的需求缺陷。当对搜索算法的时间性能进行剖析时，基于附录表 4 中的数据，可以得出以下结论：*AVM* 和*(1+1)EA* 的时间性能最好，然后是 *SSGA*，*RS* 的时间性能最差。最可能原因是平均执行时间 *RT* 十分依赖于每一对 RUCM 用况场景的相似度计算。由于 RUCM 用况规约产生的 RUCM 用况场景包含不同数量的 RUCM 需求描述语句，给定一个相似度函数， (1+1)EA 在进行相似度计算过程中的对 RUCM 需求描述语句的比较总数就有能与 AVM 的比较总数非常接近甚至一样。*SSGA* 同时还需进行交叉操作用于创建新的个体，因此要花费更多的时间。*RS* 只是随机的进行选取并没有采用任何的选择策略，因此它的搜索时间不可预估，在该实验中 *RS* 比其他搜索算法花费了更多的时间。 S^3RUCM 方法的时间耗费是现实可接受的，例如，$CSA_{NW-(1+1)EA}$用时 177 秒返回的 RUCM 用况场景集合包含了预先注入的全部需求缺陷。

5.4.7 有效性风险的讨论

5.4.7.1 构建风险分析

在该实证研究中，设计并采用有效性评价指标：*FV*、*DDR*、*EFF*，这些评价指标可以在不同的 *CSA* 实例之间进行比较。此外，所有搜索算法都采用相同的结束策略，即 25000 次的比较。在实验研究中有一个受控的成本指标：*NUCS*。对每一个 RUCM 用况规约ucs， 针对每一个 *NUCS* 取值(即 10%, 20%, 30%...90%)进行独立的实验，并收集相应的实验数据进行统计分析，从而判定 *NUCS* 对有效性指标的影响。

一个影响该实证研究的有效性的风险是：采用 RUCM 变异技术进行手工的故障注入时可能会引入额外的人为的需求缺陷。在具体的实验过程中，针对工业案例的研究，一位来自工业合作方的负责相关案例的工程师协助完成相关的实验。经过具体的培训，该工程师创建了工业案例的初始的 RUCM 模型，然后作者从遵循 RUCM 限制规则的角度出发，对创建的 RUCM 模型进行修正，并且每次修正都得到工业工程师的确认。因此，可以认为创建的 RUCM 模型并不存在额外未知的需求缺陷。针对其他的文献案例，作者非常仔细的对其采用 RUCM 进行描述，而未改变其原始的语义。这些文献案例早已用作 RUCM[8,23]的评估案例。此外，在进行故障注入时，作者严格遵循 RUCM 变异指导规则(第 4 章)从而最大程度的避免额外需求缺陷的意外引入。

5.4.7.2 内部风险分析

考虑到相似度函数和搜索算法在该实验中的使用，它们的参数设置可能会影响 S^3RUCM 的成本-效益。实验中依据相关的文献进行标准的参数设置(见 5.4.3.6 节)。该实验并未进行参数精调，因为已有文献研究[70]和[145]证明目前采用的参数设置可以对各种不同的软件工程问题进行成本-效益分析。另外，对相似度函数和搜索算法实施参数精调将会大大增大实证研究的实验规模，需要进行大量的实证分析，这是本文的一项未来工作。

该实验中采用了来自不同领域的 21 个用况，它们的复杂程度是不受控制的，因此无法实现对实验对象的随机选择。由于无法获取工业合作方的需求开发历史记录，因此也无法对注入的需求故障进行针对性的控制。该实验无法验证实验设计中的自由变量和依赖变量之间的关系。

5.4.7.3 结论风险分析

该实验遵循严格的统计分析过程对实验结果数据进行统计分析。Vargha-Delaney 统计方法、Wilcoxon 秩和检验同时搭配 Bonferroni 修正方法、Kendall's Tau 检验在实验中被使用。其中 Bonferroni 修正用法用于对多重比较中 Wilcoxon 秩和检验返回的 p-value 进行修正以防止第 I 类统计错误。影响结论有效性的最大风险是：实验中存在随机算法的随机性。为了解决这个风险，针对每个CSA_i，令其独立执行 100 次并将获取的 100 个结果数据作为统计分析的样本数据，然后进行统计分析；此外，Wilcoxon 秩和检验和 Bonferroni 修正方法搭配使用，对返回的 100 次 *FV* 实例和 100 次 *DDR* 实例进行统计分析，以判定相应的统计差异。另外，根据文献[131]的指导，Vargha-Delaney 统计的$\hat{A}_{12}$效应因子与 Wilcoxon 秩和检验一起来对统计结果进行更好的解释说明。

5.4.7.4 外部风险分析

有关软件工程实验的一个通常的外部风险就是：实验结论的通用性。该实验中采用了一个工业案例[102]，人们可能会争论说其实验结果不适用与其他的案例。然而，这样的外部风险是进行实证研究非常普遍的。此外，除了工业案例，该实验还采用了 6 个不同的文献案例用以对 S^3RUCM 进行成本-效益分析。实验中采用 *FV*、*DDR*、*EFF* 这些有效性评价指标，它们可以在不同的 *CSA* 实例间进行比较。所有的搜索算法都采用相同的停止标准，即 25000 次比较。在实验中，无论是文献案例还是工业案例，它们都显示一致的结论。

5.5 相关研究工作的对比

5.5.1 用况评审技术

自 1976 年 Dr. Fagan[15]首次阐述了软件程序评审的正式流程，软件评审一直被用作确保软件质量的一种有效技术[173]。关于需求评审的文献综述被研究学者相继发表。在文献综述[146]中，作者分别从需求评审技术、需求评审对工程项目的效益、需求评审活动的组织管理、需求评审支持工具四个视角对选取的 120 个一手研究资料进行分析。文献综述[137]的研究涵盖：Fagan 评审方法、评审过程的改进(如 N 重评审[174])、评审支持技术 (如评审阅读技术、评审工具)。文献综述研究[175]，选取了 1980-2008 年间发表的

153 篇有关需求评审的研究文章进行研究，并指出 1993 年前的研究主要集中在评审过程的研究，而很少关注评审阅读技术的研究。

在软件工程实践中，识别需求文档中的缺陷是软件质量保证的一种切实高效的技术[135]。随着用况建模技术在工业实践的广泛使用[5]，用况评审成为需求评审研究中的一个新的热点。针对用况评审，无论是研究学者还是工业实践，基于检查列表的阅读技术(Checklist-based Reading (CBR)[147,11,148])h 基于视角的阅读技术(Perspective-based Reading (PBR)[17]) 是最常用的用况评审阅读技术。

在文献[176]中，作者针对用况模型提出一种缺陷分类并设计相应的检查列表对用况模型进行缺陷识别。作者分别从系统用户视角、项目管理者视角、系统设计者视角、系统测试者视角对用况缺陷类型进行定义，并从模型缺失、模型不正确、不一致、模糊性、额外信息5个层面对用况模型构成元素进行分析。作者将提出的CBR技术与Ad-hoc阅读技术进行受控实验，实验数据显示基于检查列表的阅读技术，对于那些有着丰富经验或者对需求缺陷类型特别熟悉的评审员的帮助并不大。

Cox[6]等人提出一种针对用况规约的CBR评审技术。 根据用况规约中出现的错误的影响性，作者设计的用况规约检查列表分为三个组成部分：(1) 轻微影响：需求描述中使用了不正确的语句时态。(2) 需求规约层面的影响：抽象层次的不一致(需要把问题域和系统内部设计进行区分)。 (3) 对真实需求的影响：侧重于会引起对用户需求误解的错误。例如，用况规约的不正确描述，事件流中的错误的语句顺序。作者将他们提出的CBR技术与Ad-hoc技术进行对比，实验结果显示文章提出的CBR技术要比Ad-hoc技术显著得好，它能够帮助评审人员发现更多的语义错误。值得注意的是，Cox[6]等还定义了6种启发思路可以作为对用况规约进行验证的度量指标，在我们的研究工作[177]中对这6种启发思路进行了详细讨论。随后，文献[117] 基于文本理解理论对这6种启发思路进一步发展，提出了“7 Cs”[117]对用况规约中可能出现的语义错误进行分类，并可以剪裁到具体的用况规约模板中。

Bernárdez[178]等提出了一种基于度量的阅读技术(Metric-based Reading (MBR))，他们针对用况模型中的用况设计了相应的度量指标：*NOAS* (一个用况中参与者Actor的所有动作语步)，*NOSS* (一个用况中所有的系统动作语步)。MBR背后的假设是：如果一个度量指标的数值太低或太高，可能会识别出相应的缺陷。作者将MBR和CBR进行对比实验，

实验结果显示：在有效性(缺陷发现率)方面，MBR要比CBR显著得好；而在效率方面，两者没有显著差异。

在文献[116]中，作者展示了一种集成用况创建和用况评审的综合方法，并依据标准IEEE STD[72]定义了用况缺陷分类。作者提出一种PBR技术，其中PBR由六个视角构成：测试者视角、设计者视角、系统用况视角、系统维护视角、领域专家视角、项目管理者视角。他们设计了一个受控实验对提出的PBR和CBR进行对比分析，实验结果显示针对复杂系统，文献提出的PBR技术要比CBR显著得好。

文献[179]中，作者提出了一种CBR技术用以对用况规约进行评审，该方法已经IEEE std. 830-1998[72] 定义缺陷分类(即 *Ambiguousness*, *Incorrectness*, *Inconsistency*, *Incompleteness*)。作者将设计的CBR与PBR技术进行对比实验，实验由来自产品生产公司的开发人员进行，最后的实验结果显示：在发现缺陷的数量方面：PBR表现的比CBR显著得好，然而在效率和误报率方面：CBR表现的要比PDB显著得好。

有些研究学者关注用况缺陷识别的自动化技术研究。Sinha[180]等人提出一种方法在用况编辑时进行自动化的语义分析进行用况缺陷识别(如缺少参与者)。Fantechi[181]等借助自然语言的语义分析技术自动识别用况规约中的模糊描述和费解描述。

表34总结了各种不同的用况评审技术，从中可以发现：(1) CBR和PBR是进行用况评审中最常用的两种技术，(2) 用况评审非常依赖于用况模型的缺陷分类定义。

表 34 不同用况评审技术的特性

	Reading Technique	Artifact under inspection	Defect Type	Evaluation
[176]	CBR with a predefined defect taxonomy of use case models	Use case models (and use case specification with a simple format (i.e., flow of events, trigger, pre-/post-condition, alternatives))	Syntactic and semantic	Controlled experiment
[6]	CBR with a predefined defect taxonomy of use case specifications	Use case specifications with structured template [182]	Semantic	Controlled experiment
[178]	MBR	Use case specification with structured template [183]	Semantic	Controlled experiment
[116]	PBR based on a defect taxonomy conforming to the IEEE 830-1998 standard	Use case models (use case diagram and structured use case template)	Syntactic and semantic	Controlled experiment
[179]	CBR, PBR	Use case specifications in Use-Case 2.0 format [184]	Syntactic and semantic	Quasi-experiment

与上述研究不同，本文提出的用况场景选择方法依赖于相似度计算函数和搜索算法。本文并不从具体的评审技术的设计角度进行需求评审的工作支撑，而从如何选取评审源为评审工作提供输入的角度进行需求评审的支撑。需求评审技术发展到现在已经经历了40年的历史，目前的软件开发已发生了显著的变化，如面向对象开发技术和模型驱动技术背后的哲学思想相比较之前的结构化开发已发生巨大变化。软件规模更加庞大和复杂，软件密集型系统(software-intensive system)、系统集成系统(system of systems)在日常生活中成为主要的服务提供者，尤其是航空工业领域。此外，在市场驱动的开发环境下，需求评审的开展必须要采用一种成本-效益的方式进行，因此本文从评审源选择的角度研究需求评审的支撑技术。如5.2.1节所论述的，RUCM提供结构化的需求描述模板对用况场景进行组织管理，因此对RUCM用况规约的评审最终都会转化为对用况场景的评审。此外，文献[176,6,179,117]中提出的用况评审方法实际上也是对用况场景的评审，因此本文提出的用况场景选择方法可以与不同的评审技术(CBR、PBR)结合使用。

5.5.2 基于搜索的需求选择

Harman[56]等对基于搜索的软件工程的研究进行了全面系统的文献综述，并指出不同的搜索算法被应用于解决软件工程中的各种问题，例如需求选择和优化问题[57]、需求分配问题[58]、需求优先化的问题[59]。NRP问题(The Next Release Problem (NRP)[57])是一个典型的使用搜索技术解决的需求选择问题，即通过选择一部分需求实现资源约束和用户需求之间的平衡。文献[57]使用了五个不同规模的问题和三类不同的搜索策略(精确技术、贪心算法、局部搜索(爬山算法、模拟退火算法))对NRP进行阐述，其中最小规模的问题有100个客户和140个任务，最大规模的问题有500个客户和3250个任务。实验结果显示：精确技术对于解决最小规模的问题性能显著好，对于规模较大的问题，模拟退火算法的性能显著的好。Baker[59]等采用贪心算法和模拟退火算法解决软件构件选择的NRP问题，实验结果显示这两个算法的性能要比领域专家的判断显著得好。

NRP问题又被发展为多目标的NRP(Multi-Objective Next Release Problem (MONRP)[60])。MONRP主要被设计来挑选一些客户需求，以使得公司利益最大化同时把实现这些客户需求的成本最小化。Zhang[60]采用四种不同的搜索算法研究MONRP：随机搜索、单目标遗传算法、帕累托遗传算法、非优势排列遗传算法。实验结果显示：非优势排列遗传算法比较适合于解决MONRP问题。Saliu和Ruhe[61]提出一种决策支持方法将

NRP问题定义为一种双目标优化问题，并采用ε-约束算法[62]对双目标优化问题进行求解。在文献[63]中，作者采用演化算法和量子计算研究MONRP问题。 文献[64]中，作者采用蚁群算法(ACO)解决NRP，他们将ACO与贪婪随机适应搜索策略(GRASP)、非优势排列遗传算法(NSGA)进行对比。实验结果显示：GRASP的性能最差， NSGA难以使用，ACO可以有效的解决NRP。

Greer和Ruhe[65]提出一种软件发布计划(RP)问题。RP旨在选择和分配一些新的特性并对需求进行修改从而支持一系列连续的产品发布计划。RP主要研究两方面的内容：哪些需要发布(what)和什么时候发布(when)[61]。Li[66]等提出一种集成需求选择和分配的方法解决RP问题，他们采用01背包模型[67]对需求选择进行解决。文献[68]中，作者将基于主题的RP问题形式化为一个双目标优化问题，并采用多目标优化算法NSGA-II[69]进行解决。

Li[70]等针对CPS系统(Cyber-Physical Systems)提出一种需求分配的方法，该方法在将需求分配给不同利益相关者时，实现分配者对需求熟悉程度的最大化同时平衡相应的工作量。她们分别设计单目标和多目标的优化策略，并选用一个工业案例和120个人工问题进行实验。实验结果显示：对于单目标优化问题的解决，(1+1) EA的性能最好， 而在多目标优化的问题解决中NSGA-II的性能最好。

与当前的研究相比，本章提出的优化方法属于基于搜索的软件工程中的需求选择这类应用，本章提出的优化目标是对选择的用况场景的平均相似度的最小化，从而可以支持成本-效益的需求评审。

5.5.3 相似度函数在软件工程中的应用

相似度计算函数已被应用到软件测试中，特别是测试用例筛选问题。在文献[144]中，作者提出一种基于相似度的测试用例优先化的方法。在该方法中测试套件(即不同的测试用例)被抽象为字符串，然后借助相似度函数(*Hamming*、*Levenshtein*、*Cartesian*、*Manhattan*)和贪心算法对测试用例进行排序，从而实现测试用例的优先化排序。

Hemmati[145]等使用一系列相似度计算函数和搜索算法实现了可扩展的模型驱动测试。该方法使用相似度函数对测试用例进行选择：(1) 对测试用例进行抽象编码，(2) 设定相似度度量指标用于对每一对抽象编码后的测试用例计算相应的相似度数值，(3) 采用搜索算法对相似度数值进行最小化。在提出的测试用例选择方法中，作者使用了基于

集合的相似度函数(*Hamming, Jaccard, Dice, Gower-Legendre, Counting*)和基于序列的相似度函数(*Levenshtein, Needleman-Wunsch, Smith-Waterman*)。作者选用了两个工业案例进行实验，实验数据显示：*Gower-Legendre*相似度函数和 (1+1) EA组合是最符合成本-效益的测试用例选择方法。

与上述研究[144,145]相比，本文采用相似度函数解决用况场景的选择问题，即使用相似度函数(CNT、JAC、GOW、SOK、NLCS、LEV、NW、SW)对每一对用况场景进行相似度计算，然后再使用搜索算法(AVM、SSGA、(1+1)EA、RS)对用况场景的平均相似度数值进行最小化的选择。

5.6 本章小结

需求在大型复杂实时系统的开发中扮演关键的角色。由于大型系统的先天复杂性，这类系统的开发过程中大量的精力会投入到需求工程中，因为需求的质量直接影响后续的开发活动。随着用况建模技术作为捕获、描述需求的一种技术在工业实践中被广泛使用，用况评审得到研究学者和工业实践者的高度关注。在大型复杂实时系统的背景下，大量的用况会导致大量的用况场景，而这些用况场景通常都是需求评审的评审源。在有限的评审成本下，如何选择一部分用况场景支持成本-效益的手工评审是一个重要的优化选择问题。

本章借鉴相似度计算函数和搜索算法设计一种用况场景选择的优化方法S^3RUCM。S^3RUCM由四种搜索算法：*(1+1) EA, SSGA, AVM* and the *R*和八种相似度计算函数：*CNT, JAC, GOW, SOK, NLCS, LEV, NW, SW*构成。为了指导搜索算法的而有效选择，本文根据用况场景选择问题设计了相应的适应度函数。本章采用一个工业案例和六个文献案例对提出的S^3RUCM进行实证研究。为了验证不同搜索算法的性能，即选择出一组平均相似度低但缺陷发现率高的用况场景组合，本章设计了一系列的实验进行研究，实验结果的统计分析显示：本章选择的搜索算法*(1+1) EA, SSGA, AVM* 都要比随机搜索RS要显著得好；在所有的搜索算法中*(1+1) EA*的性能最好。

由于S^3RUCM由四种搜搜算法和八种相似度计算函数构成，本章有针对这32种不同的组合设计相应的实验进行评价分析。实验数据的统计分析显示：*NW*和*(1+1) EA*组合的性能最好，因为它可以选择出一半的用况场景发现所有注入的需求缺陷。为了进一步研

究相似度数值和缺陷发现率之间的关系，实验中采用Kendall's[167]Tau统计检验对适应度函数值*FV*和缺陷发现率*DDR*进行分析，统计分析显示：在RUCM用况模型的环境下，*FV* (代表相似度的)和*DDR* (缺陷发现率)之间存在着显著的负相关性。基于以上结果，本文推荐*(1+1) EA*和*NW*组合进行工业实践。

结 论

论文总结

软件需求是软件开发后续活动(分析、设计、测试)的依据和溯源。针对复杂实时系统的需求开发和需求评审，本文从四个方面开展研究工作：基于 RUCM 的实时系统用况建模，面向方面的实时系统用况建模，基于工业标准的用况变异分析，基于搜索算法的用况场景选择。通过选用典型的工业案例和文献案例对论文提出的方法进行验证分析。本文的主要贡献：

1) 提出一种基于 RUCM 的实时系统用况建模方法

本文对 MARTE[71]外廓(UML profile for Modeling and Analysis of Real-Time and Embedded Systems)实时系统建模指导规范进行系统化研究，通过扩展 RUCM[23,8,24] (Restriced Use Case Modling)用况建模方法，提出实时系统的用况建模方法 RUCM4RT，从而辅助需求人员对实时特性需求进行识别和规约。RUCM4RT 继承了 RUCM 优秀的用况规约能力，为需求开发阶段提供重要的制品用况规约。此外 RUCM4RT 还会自动生成 UML 定时图(Timing Diagram)，从而辅助需求人员进行有限的分析。论文分别采用航空领域和航海领域的两个工业案例对 RUCM4RT 方法进行验证分析，实验结果显示：RUCM4RT 可以系统化的帮助需求人员对各种不同的实时用况进行识别和规约。

2) 提出一种面向实时特性需求的用况建模方法

针对复杂实时系统的用况建模，本文提出了一种 RUCM 的扩展，即 rtAspectRUCM，它在用况模型层次实现了横切关注点的建模，从而可以辅助 RUCM4RT 对大型复杂的实时系统进行需求建模。rtAspectRUCM 采用了一个 UML 外廓将横切关注点建模为用况图和用况规约中的切面模型。rtAspectRUCM 扩展对于复杂实时系统的用况建模是必要的，在这类系统的用况建模中需要对额外的非功能需求进行捕获和规约。为了对 rtAspectRUCM 的有效性(即节省建模工作量)进行评估，本文选用了三个工业案例进行实验分析。实验结果显示，针对横切关注点 rtAspectRUCM 平均可以节省 80%的建模工作量。

3) 提出一种基于工业标准的用况变异分析方法

针对不同的需求评审方法的有效评估，在目前的学术研究中存在两个明显的问题：1) 缺少一种系统化的创建用况模型缺陷的方法，从而无法全面的评价不同的需求评审

技术的有效性；2) 需要一种针对用况模型的全面的缺陷分类，并且针对每一个缺陷类型要给出确切的定义。这样才可以准确的定义缺陷发现率从而对不同的评审技术进行有效的评估。本文提出一种变异分析方法 MuRUCM 用以解决上述问题。通过对工业标准 IEEE Std. 830-1998 进行系统化的研究，定义了一个针对用况模型的需求缺陷分类。通过严格遵循工业标准 HAZOP 对 RUCM 用况模型进行分析，创建了一系列的 RUCM 变异算子用以创建不同类型的需求缺陷实例。最后，给出了一组辅助规则用以创建缺陷生成策略。本章从不同的领域选用了不同复杂程度的 2 个工业案例和 9 个文献案例对提出的 MuRUCM 方法进行验证。实验结果显示：提出的需求缺陷分类适用于 RUCM 用况模型，生成的 RUCM 变种能够有效地区分三种不同的 RUCM 用况场景生成策略。

3) 提出一种基于相似度和搜索算法的用况场景选择方法

在大型复杂实时系统的背景下，大量的用况会导致大量的用况场景，而这些用况场景通常都是需求评审的评审源。在有限的评审成本(时间、人力)下，如何选择一部分用况场景支持成本-效益的手工评审是一个重要的优化选择问题。

本文借鉴相似度计算函数和搜索算法，设计一种用况场景选择的优化方法S^3RUCM。S^3RUCM由四种搜索算法：*(1+1) EA*、*SSGA*、*AVM* 、*RS*和八种相似度计算函数：*CNT*、*JAC*、*GOW*、*SOK*、*NLCS*、*LEV*、*NW*、*SW*构成。为了验证不同搜索算法的性能，即选择出一组平均相似度低但缺陷发现率高的用况场景组合，本文设计了一系列的实验进行实证研究。实验结果的统计分析显示：本文选择的搜索算法*(1+1) EA*、*SSGA*、*AVM* 都要比随机搜索*RS*显著得好；在所有的搜索算法中*(1+1) EA*的性能最好。本文针对这32种不同的组合设计相应的实验进行评价分析，实验数据的统计分析显示：*NW*和*(1+1) EA*组合的性能最好，因为它可以选择出一半的用况场景发现所有注入的需求缺陷。为了进一步研究相似度数值和缺陷发现率之间的关系，实验中采用Kendall's[167]Tau统计检验对适应度函数值*FV*和缺陷发现率*DDR*进行分析，统计分析显示：在RUCM用况模型的环境下，*FV* (代表相似度的)和*DDR* (缺陷发现率)之间存在着显著的负相关性。基于以上结果，本文推荐*(1+1) EA*和*NW*组合进行工业实践。

进一步工作

本文的后续研究工作包括以下三个方面：

1) 在案例研究方面，目前本文提出的方法采用工业案例和文献案例进行验证评估。

在后续工作中，将收集更多实时领域的案例进行验证并完善本文提出的上述方法。

2) 针对实时系统的用况建模，第 3 章提出的 RUCM4RT 方法能够把生成的实时用况场景自动生成 UML 定时图从而辅助需求人员进行有限的分析。在后续的工作中将引入模型执行的方法，将建模后的用况模型自动转换成 UML 活动图，然后通过活动图的执行实现对实时约束的动态验证。

3) 针对用况评审的支撑，第 4 章和第 5 章分别实现了用况缺陷的系统化注入和用况场景的择优选择。在后续的研究工作中，将针对 RUCM 用况模型研究具体的用况评审方法，从而完善 RUCM 用况评审的整个方法体系。

附 录

第五章的实验详细结果

RQ1 的统计分析结果

为了回答 RQ1，给定每一个相似度函数，附录表 1 分别以为评价指标 *DDR* (5.4.3.3 节) 和 *FV* (5.4.3.3 节)对每一个搜索算法与随机搜索进行统计分析。A>B 表示在所有优化问题中搜索算法 A 比搜索算法 B 显著性好的总数， A<B：搜索算法 B 比搜索算法 A 显著性好的总数，A=B：两者之间不存显著性差异(p-value > 0.05)的总数。

附录表 1 RQ1 的实验结果，使用了 Vargha-Delaney 统计，Wilcoxon 秩和检验和 Bonferroni 修正方法，显著水平为 0.05

Similarity Function	Pair of Algorithms (A vs. B)	Statistical Test Result											
		FV			*DDR*			*FV*			*DDR*		
		A > B	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B
		NUCS = 10%						*NUCS = 20%*					
CNT	*SSGA vs. RS*	16	2	3	13	3	5	21	0	0	13	4	4
	EA vs. RS	12	5	4	10	4	7	17	4	0	13	4	4
	AVM vs. RS	16	2	3	14	2	5	16	2	3	15	1	5
JAC	*SSGA vs. RS*	16	2	3	12	4	5	21	0	0	13	6	2
	EA vs. RS	12	5	4	10	4	7	16	4	2	12	5	4
	AVM vs. RS	16	2	3	14	1	6	14	4	2	15	1	5
GOW	*SSGA vs. RS*	16	1	4	13	4	4	21	0	0	13	3	5
	EA vs. RS	13	6	2	10	4	7	17	4	0	12	4	5
	AVM vs. RS	16	1	4	14	2	5	17	3	1	15	1	5
SOK	*SSGA vs. RS*	16	2	3	8	5	8	21	0	0	11	4	6
	EA vs. RS	12	6	4	10	5	6	14	2	5	12	5	4
	AVM vs. RS	16	2	3	12	2	7	15	2	4	14	1	6
NLCS	*SSGA vs. RS*	16	3	2	10	5	6	21	0	0	12	2	7
	EA vs. RS	12	6	3	12	3	6	16	4	1	10	4	7
	AVM vs. RS	16	3	2	13	4	7	18	1	2	15	1	5
LEV	*SSGA vs. RS*	14	0	7	12	1	8	21	0	0	12	4	5
	EA vs. RS	13	5	3	12	4	5	17	2	2	10	6	5
	AVM vs. RS	14	0	7	10	4	7	15	2	4	14	1	6
NW	*SSGA vs. RS*	14	0	7	15	0	6	21	0	0	18	1	2
	EA vs. RS	12	5	4	17	0	4	16	3	2	17	1	3
	AVM vs. RS	14	0	7	17	1	3	19	0	2	20	0	1

附录表 1-续 1

Similarity Function	Pair of Algorithms (A vs. B)	Statistical Test Result											
		FV			*DDR*			*FV*			*DDR*		
		A > B	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B
		NUCS = 10%						*NUCS = 20%*					
SW	*SSGA vs. RS*	14	0	7	13	0	8	21	0	0	18	2	1
	EA vs. RS	11	5	5	17	1	3	16	2	3	17	1	3
	AVM vs. RS	14	0	7	19	1	1	19	0	2	20	0	1
		NUCS = 30%						*NUCS = 40%*					
CNT	*SSGA vs. RS*	20	0	1	15	4	2	21	0	0	15	4	2
	EA vs. RS	19	2	0	14	3	4	20	0	1	15	2	4
	AVM vs. RS	15	2	3	14	2	5	14	1	6	15	0	6
JAC	*SSGA vs. RS*	21	0	0	15	1	5	21	0	0	17	2	2
	EA vs. RS	18	1	2	13	2	6	20	0	1	16	2	3
	AVM vs. RS	16	1	4	14	1	6	14	2	5	15	0	6
GOW	*SSGA vs. RS*	20	0	1	15	2	4	21	0	0	19	1	1
	EA vs. RS	19	2	0	13	3	5	21	0	0	18	1	2
	AVM vs. RS	15	2	4	15	2	4	13	2	6	17	0	4
SOK	*SSGA vs. RS*	21	0	1	15	1	5	21	0	0	16	3	2
	EA vs. RS	19	0	2	16	2	3	21	0	0	17	1	3
	AVM vs. RS	15	2	4	14	2	5	16	1	4	16	0	5
NLCS	*SSGA vs. RS*	20	0	1	16	4	1	21	0	0	13	2	6
	EA vs. RS	18	2	1	14	4	3	20	0	1	17	1	3
	AVM vs. RS	15	1	5	15	2	4	15	1	5	16	0	5
LEV	*SSGA vs. RS*	20	0	1	11	6	4	21	0	0	13	2	6
	EA vs. RS	18	2	1	11	5	5	21	0	0	14	3	4
	AVM vs. RS	15	1	5	15	0	6	15	2	4	17	0	4
NW	*SSGA vs. RS*	21	0	0	19	2	0	21	0	0	19	2	0
	EA vs. RS	20	0	1	19	1	1	20	0	1	18	1	2
	AVM vs. RS	20	1	0	19	1	1	19	0	2	19	0	2
SW	*SSGA vs. RS*	21	0	0	19	2	0	21	0	0	19	2	0
	EA vs. RS	20	0	1	20	1	0	20	0	1	19	1	1
	AVM vs. RS	20	1	0	19	0	2	19	0	2	19	0	2
		NUCS = 50%						*NUCS = 60%*					
CNT	*SSGA vs. RS*	21	0	0	17	3	1	21	0	0	20	0	1
	EA vs. RS	21	0	0	17	3	1	21	0	0	20	0	1
	AVM vs. RS	17	3	1	16	1	4	14	4	3	16	0	5
JAC	*SSGA vs. RS*	21	0	0	17	1	3	21	0	0	20	1	0
	EA vs. RS	21	0	0	18	2	1	21	0	0	20	1	0
	AVM vs. RS	15	1	5	14	1	6	14	3	4	16	0	5

附录表 1-续 2

Similarity Function	Pair of Algorithms (A vs. B)	Statistical Test Result											
		FV			*DDR*			*FV*			*DDR*		
		A > B	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B
		NUCS = 50%						*NUCS = 60%*					
GOW	*SSGA vs. RS*	21	0	0	17	2	2	21	0	0	20	0	1
	EA vs. RS	21	0	0	17	3	1	21	0	0	20	0	1
	AVM vs. RS	15	3	3	13	1	7	14	3	4	17	0	4
SOK	*SSGA vs. RS*	21	0	0	17	1	3	21	0	0	20	1	0
	EA vs. RS	21	0	0	17	2	2	21	0	0	20	1	0
	AVM vs. RS	16	1	4	15	1	5	15	3	3	16	0	5
NLCS	*SSGA vs. RS*	21	0	0	16	3	2	21	0	0	18	0	3
	EA vs. RS	21	0	0	16	3	2	21	0	0	19	1	1
	AVM vs. RS	15	3	3	15	1	5	13	3	5	14	0	7
LEV	*SSGA vs. RS*	21	0	0	17	4	0	21	0	0	16	4	1
	EA vs. RS	21	0	0	16	5	0	21	0	0	16	5	0
	AVM vs. RS	16	2	3	14	2	5	14	3	4	16	0	5
NW	*SSGA vs. RS*	21	0	0	19	0	2	21	0	0	20	0	1
	EA vs. RS	21	0	0	18	1	2	21	0	0	20	0	1
	AVM vs. RS	19	1	1	18	0	3	15	1	5	17	0	4
SW	*SSGA vs. RS*	21	0	0	17	1	3	21	0	0	20	1	0
	EA vs. RS	21	0	0	18	0	3	21	0	0	20	1	0
	AVM vs. RS	18	1	2	18	0	3	13	2	6	18	1	2
		NUCS = 70%						*NUCS = 80%*					
CNT	*SSGA vs. RS*	21	0	0	19	0	2	21	0	0	21	0	0
	EA vs. RS	21	0	0	19	0	2	21	0	0	21	0	0
	AVM vs. RS	12	6	3	16	2	3	14	2	5	16	0	5
JAC	*SSGA vs. RS*	21	0	0	20	0	1	21	0	0	21	0	0
	EA vs. RS	21	0	0	20	0	1	21	0	0	21	0	0
	AVM vs. RS	13	5	3	14	2	5	15	1	5	17	0	4
GOW	*SSGA vs. RS*	21	0	0	19	0	2	21	0	0	21	0	0
	EA vs. RS	21	0	0	19	0	2	21	0	0	21	0	0
	AVM vs. RS	13	5	3	17	2	2	14	1	6	17	0	4
SOK	*SSGA vs. RS*	21	0	0	18	1	2	21	0	0	21	0	0
	EA vs. RS	21	0	0	18	1	2	21	0	0	21	0	0
	AVM vs. RS	13	5	3	16	2	3	15	1	5	17	0	4
NLCS	*SSGA vs. RS*	21	0	0	18	2	1	21	0	0	21	0	0
	EA vs. RS	21	0	0	18	2	1	21	0	0	21	0	0
	AVM vs. RS	14	6	1	14	2	5	14	1	6	19	0	2

附录表 1-续 3

Similarity Function	Pair of Algorithms (A vs. B)	Statistical Test Result											
		FV			*DDR*			*FV*			*DDR*		
		A > B	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B
		NUCS = 70%						*NUCS = 80%*					
LEV	*SSGA vs. RS*	21	0	0	16	3	1	21	0	0	21	0	0
	EA vs. RS	21	0	0	16	4	1	21	0	0	21	0	0
	AVM vs. RS	11	6	4	14	2	5	15	1	5	19	0	2
NW	*SSGA vs. RS*	21	0	0	19	0	2	21	0	0	21	0	0
	EA vs. RS	21	0	0	19	0	2	21	0	0	21	0	0
	AVM vs. RS	13	6	2	14	2	5	14	2	5	18	0	3
SW	*SSGA vs. RS*	21	0	0	19	0	2	21	0	0	21	0	0
	EA vs. RS	21	0	0	19	0	2	21	0	0	21	0	0
	AVM vs. RS	11	5	5	18	2	1	13	3	5	18	0	3
		NUCS = 90%											
CNT	*SSGA vs. RS*	21	0	0	21	0	0						
	EA vs. RS	21	0	0	21	0	0						
	AVM vs. RS	15	1	5	18	0	3						
JAC	*SSGA vs. RS*	21	0	0	21	0	0						
	EA vs. RS	21	0	0	21	0	0						
	AVM vs. RS	15	1	5	18	0	3						
GOW	*SSGA vs. RS*	21	0	0	21	0	0						
	EA vs. RS	21	0	0	21	0	0						
	AVM vs. RS	15	1	5	17	0	4						
SOK	*SSGA vs. RS*	21	0	0	21	0	0						
	EA vs. RS	21	0	0	21	0	0						
	AVM vs. RS	15	0	6	17	0	4						
NLCS	*SSGA vs. RS*	21	0	0	21	0	0						
	EA vs. RS	21	0	0	21	0	0						
	AVM vs. RS	15	0	6	18	0	3						
LEV	*SSGA vs. RS*	21	0	0	21	0	0						
	EA vs. RS	21	0	0	21	0	0						
	AVM vs. RS	15	2	4	19	0	2						
NW	*SSGA vs. RS*	21	0	0	21	0	0						
	EA vs. RS	21	0	0	21	0	0						
	AVM vs. RS	15	0	6	17	0	6						
SW	*SSGA vs. RS*	21	0	0	21	0	0						
	EA vs. RS	21	0	0	21	0	0						
	AVM vs. RS	15	2	4	16	0	5						

RQ2 的统计分析结果

为了回答 RQ2，给定每一个相似度函数，附录表 2 和附录表 3 分别以为评价指标 *DDR* (5.4.3.3 节) 和 *FV* (5.4.3.3 节)对每一对搜索算法(A *vs.* B)进行统计分析。A>B 表示在所有优化问题中搜索算法 A 比搜索算法 B 显著性好的总数， A<B：搜索算法 B 比搜索算法 A 显著性好的总数，A=B：两者之间不存显著性差异(p-value > 0.05)的总数。

附录表 2 RQ2 的统计分析结果(*DDR*)，使用了 Vargha-Delaney 统计，Wilcoxon 秩和检验和 Bonferroni 修正方法，显著水平为 0.05

SIM	*Pair of search algorithms (A vs. B)*	*NUCS*														
		10%			*20%*			*30%*			*40%*			*50%*		
		A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B
CNT	*SSGA vs. AVM*	3	8	10	8	7	6	9	6	6	11	5	5	16	3	2
	SSGA vs. EA	6	5	10	5	7	9	6	8	7	4	9	8	4	7	10
	AVM vs. EA	9	5	7	9	8	4	6	9	6	3	11	7	2	16	3
JAC	*SSGA vs. AVM*	6	5	10	10	7	4	10	4	7	12	3	6	15	2	4
	SSGA vs. EA	6	4	11	6	6	9	3	6	12	2	8	11	6	9	6
	AVM vs. EA	7	3	11	9	8	4	3	8	10	2	13	6	1	17	3
GOW	*SSGA vs. AVM*	4	8	9	8	7	6	9	7	5	10	5	6	17	2	2
	SSGA vs. EA	7	3	11	6	7	8	6	7	8	4	8	9	6	6	9
	AVM vs. EA	10	4	7	9	6	6	7	8	6	1	13	7	2	15	4
SOK	*SSGA vs. AVM*	6	6	9	9	6	6	11	3	7	15	1	5	16	1	4
	SSGA vs. EA	5	3	13	6	4	11	7	4	10	7	8	6	5	8	8
	AVM vs. EA	5	1	15	9	7	5	3	10	8	1	16	4	0	15	6
NLCS	*SSGA vs. AVM*	3	7	11	7	7	7	6	8	7	11	4	6	16	2	3
	SSGA vs. EA	5	5	11	5	6	10	4	7	10	3	12	6	7	8	6
	AVM vs. EA	9	3	9	9	8	4	6	6	9	2	13	6	3	15	3
LEV	*SSGA vs. AVM*	10	2	9	7	5	9	7	9	5	10	5	6	15	5	1
	SSGA vs. EA	10	2	9	8	3	10	5	5	11	3	7	11	3	8	10

	AVM vs. EA	7	5	9	9	5	7	6	5	10	4	12	5	4	13	4

附录表 2-续 1

SIM	***Pair of search algorithms (A vs. B)***	***NUCS***														
		10%			***20%***			***30%***			***40%***			***50%***		
		A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B
NW	***SSGA vs. AVM***	12	5	4	14	3	4	17	2	2	16	3	2	16	1	4
	SSGA vs. EA	9	5	7	6	5	10	3	8	10	4	10	7	0	11	10
	AVM vs. EA	6	8	6	3	14	4	1	17	3	3	16	2	0	18	3
SW	***SSGA vs. AVM***	12	6	3	14	3	4	17	2	2	16	2	3	16	2	3
	SSGA vs. EA	8	5	8	8	4	9	5	8	8	3	7	11	2	8	11
	AVM vs. EA	7	8	6	3	14	4	1	17	3	2	16	3	2	16	3
		60%			***70%***			***80%***			***90%***					
		A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B			
CNT	***SSGA vs. AVM***	17	0	4	19	0	2	20	0	1	20	0	1			
	SSGA vs. EA	3	7	11	2	16	3	3	8	10	3	12	6			
	AVM vs. EA	1	17	3	0	20	1	0	20	1	0	20	1			
JAC	***SSGA vs. AVM***	17	1	3	19	0	2	20	0	1	20	0	1			
	SSGA vs. EA	2	6	13	3	8	10	2	8	11	3	11	7			
	AVM vs. EA	1	18	2	0	19	2	0	20	1	0	20	1			
GOW	***SSGA vs. AVM***	17	0	4	19	0	2	20	0	1	20	0	1			
	SSGA vs. EA	3	7	11	2	9	10	2	9	10	3	11	7			
	AVM vs. EA	0	17	4	0	20	1	0	20	1	0	20	1			
SOK	***SSGA vs. AVM***	19	1	1	19	1	1	20	0	1	20	0	1			
	SSGA vs. EA	5	7	9	2	7	12	3	9	9	1	11	9			
	AVM vs. EA	1	18	2	1	19	1	0	20	1	0	20	1			

附录表 2-续 2

SIM	*Pair of search algorithms (A vs. B)*	*NUCS*															
		10%			*20%*			*30%*			*40%*			*50%*			
		A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	
		60%			*70%*			*80%*			*90%*						
		A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B				
NLCS	*SSGA vs. AVM*	16	0	5	17	1	3	19	0	2	20	0	1				
	SSGA vs. EA	5	5	11	5	9	7	3	8	10	3	10	8				
	AVM vs. EA	1	17	3	1	17	3	0	19	2	0	20	1				
LEV	*SSGA vs. AVM*	16	3	2	14	2	5	17	1	3	20	0	1				
	SSGA vs. EA	4	8	9	6	5	10	2	8	11	3	9	9				
	AVM vs. EA	5	15	1	3	14	4	0	17	4	0	20	1				
NW	*SSGA vs. AVM*	18	1	2	19	0	2	20	0	1	20	0	1				
	SSGA vs. EA	0	9	12	0	9	12	0	10	11	0	0	21				
	AVM vs. EA	0	19	2	0	20	1	0	20	1	0	20	1				
SW	*SSGA vs. AVM*	18	0	3	19	0	2	20	0	1	20	0	1				
	SSGA vs. EA	0	18	3	0	8	13	0	12	9	0	0	21				
	AVM vs. EA	0	19	2	0	20	1	0	20	1	0	20	1				

附录表 3 RQ2 的统计分析结果(*FV*)，使用了 Vargha-Delaney 统计，Wilcoxon 秩和检验和 Bonferroni 修正方法，显著水平为 0.05

SIM	***Pair of search algorithms* (A vs. B)**	***NUCS***														
		10%			***20%***			***30%***			***40%***			***50%***		
		A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B	A>B	A<B	A=B
CNT	***SSGA vs. AVM***	15	2	4	20	0	1	20	0	1	20	0	1	20	0	1
	SSGA vs. EA	16	1	4	17	3	1	12	8	1	9	9	3	9	11	1
	AVM vs. EA	8	9	4	5	15	1	1	16	4	0	20	1	0	20	1
JAC	***SSGA vs. AVM***	15	1	5	20	0	1	20	0	1	20	0	1	20	0	1
	SSGA vs. EA	16	1	4	17	4	0	13	7	1	9	10	2	9	11	1
	AVM vs. EA	6	9	6	5	13	3	0	17	4	0	20	1	0	20	1
GOW	***SSGA vs. AVM***	13	2	6	20	0	1	20	0	1	20	0	1	20	0	1
	SSGA vs. EA	15	1	5	15	4	2	12	8	1	9	11	1	9	11	1
	AVM vs. EA	8	9	4	6	13	2	2	16	3	0	20	1	0	20	1
SOK	***SSGA vs. AVM***	15	0	6	20	0	1	20	0	1	20	0	1	20	0	1
	SSGA vs. EA	15	0	6	17	4	0	12	5	4	9	12	0	9	10	2
	AVM vs. EA	7	8	6	2	15	4	0	17	4	1	20	0	0	20	1
NLCS	***SSGA vs. AVM***	16	0	5	20	0	1	20	0	1	20	0	1	20	0	1
	SSGA vs. EA	19	0	2	17	2	2	12	6	3	9	10	2	8	11	2
	AVM vs. EA	11	8	2	7	12	2	1	17	3	0	20	1	0	20	1
LEV	***SSGA vs. AVM***	12	0	9	20	0	1	20	0	1	20	0	1	20	0	1
	SSGA vs. EA	17	0	4	15	2	4	11	8	2	10	11	0	8	11	2
	AVM vs. EA	7	11	3	3	16	2	0	19	2	0	20	1	0	20	1
NW	***SSGA vs. AVM***	11	0	10	20	0	1	20	0	1	20	0	1	20	0	1
	SSGA vs. EA	15	0	6	17	4	0	10	6	5	9	9	3	5	13	3

	AVM vs. EA	9	8	4	7	12	2	1	16	4	0	19	2	0	20	1

附录表 3-续 1

SIM	***Pair of search algorithms* (A vs. B)**	***NUCS***														
		10%			***20%***			***30%***			***40%***			***50%***		
		A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B
SW	***SSGA vs. AVM***	11	0	10	20	0	1	20	0	1	20	0	1	20	0	1
	SSGA vs. EA	15	0	6	17	4	0	12	7	2	9	9	3	5	13	3
	AVM vs. EA	9	8	4	7	12	2	3	16	2	0	20	1	0	20	1
		60%			***70%***			***80%***			***90%***					
		A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B			
CNT	***SSGA vs. AVM***	20	0	1	20	0	1	20	0	1	20	0	1			
	SSGA vs. EA	6	13	2	5	12	4	5	12	4	6	11	4			
	AVM vs. EA	0	20	1	0	20	1	0	20	1	0	20	1			
JAC	***SSGA vs. AVM***	20	0	1	20	0	1	20	0	1	20	0	1			
	SSGA vs. EA	5	12	4	4	12	5	6	12	3	6	12	3			
	AVM vs. EA	0	20	1	0	20	1	0	20	1	0	20	1			
GOW	***SSGA vs. AVM***	20	0	1	20	0	1	20	0	1	20	0	1			
	SSGA vs. EA	6	13	2	4	12	5	6	12	3	6	11	4			
	AVM vs. EA	0	20	1	0	20	1	0	20	1	0	20	1			
SOK	***SSGA vs. AVM***	20	0	1	20	0	1	20	0	1	20	0	1			
	SSGA vs. EA	7	13	1	5	11	5	6	12	3	6	11	4			
	AVM vs. EA	0	20	1	0	20	1	0	20	1	0	20	1			

NLCS	***SSGA vs. AVM***	20	0	1	20	0	1	20	0	1	20	0	1			
	SSGA vs. EA	5	13	3	6	11	4	5	13	3	6	11	4			
	AVM vs. EA	0	20	1	0	20	1	0	20	1	0	20	1			

附录表 3-续 2

SIM	***Pair of search algorithms* (A vs. B)**	***NUCS***														
		10%			***20%***			***30%***			***40%***			***50%***		
		A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B
		60%			***70%***			***80%***			***90%***					
		A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B	A> B	A< B	A= B			
LEV	***SSGA vs. AVM***	20	0	1	20	0	1	20	0	1	20	0	1			
	SSGA vs. EA	4	13	4	5	13	3	6	12	3	5	12	4			
	AVM vs. EA	0	20	1	0	20	1	0	20	1	0	20	1			
NW	***SSGA vs. AVM***	20	0	1	20	0	1	20	0	1	20	0	1			
	SSGA vs. EA	6	14	1	4	11	6	4	13	4	6	13	3			
	AVM vs. EA	0	20	1	0	20	1	0	20	1	0	20	1			
SW	***SSGA vs. AVM***	20	0	1	20	0	1	20	0	1	20	0	1			
	SSGA vs. EA	4	13	4	0	12	9	5	13	3	6	11	4			
	AVM vs. EA	0	20	1	0	20	1	0	20	1	0	20	1			

RQ6 的统计分析结果

附录表 4 针对每一个 *NUCS* 实例，展示了每一个 *CSA* 实例的平均执行时间。

附录表 4 每一个 *CSA* 实例的平均执行时间(时间单位：秒)

CSA		***NUCS***								
Search Algorithm	**Similarity Function**	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
(1+1) EA	***CNT***	2.9638	12.3338	26.8323	46.4225	86.3774	114.4412	151.1758	217.9833	270.6795
	JAC	3.4378	14.4303	32.5498	54.1199	99.722	132.7128	178.8165	250.6386	300.6769
	GOW	3.4406	14.2663	32.1866	53.7792	99.4069	131.9904	183.9755	248.3777	300.0031
	SOK	3.5055	13.8671	31.4575	53.9228	99.1807	130.3091	175.1098	242.6777	301.7805
	NLCS	5.8833	25.2622	59.5904	100.1396	183.2946	254.2974	352.3153	479.6237	638.9036
	LEV	6.7654	26.5397	61.7045	104.6837	183.1819	250.0322	346.2966	468.3714	640.6613
	NW	5.5026	24.1807	56.6726	98.864	176.5596	235.9053	345.738	460.6211	628.0258
	SW	6.3649	27.6047	64.5336	112.158	205.0673	276.6232	383.8888	496.1493	687.4767
AVM	***CNT***	2.6668	11.3628	24.9684	49.3178	86.2911	116.7713	161.2557	221.1263	284.6687
	JAC	2.9734	13.8219	29.5663	57.9545	104.5442	133.834	189.0433	260.7807	325.377
	GOW	2.9365	13.6593	29.687	57.4548	98.849	133.101	186.0246	260.0776	322.4497
	SOK	2.9942	13.2582	29.7549	57.1215	97.1635	132.2748	185.6814	252.7304	319.8351
	NLCS	5.2578	22.5369	53.5463	105.5656	177.2594	257.59	355.2665	490.4371	615.2067
	LEV	6.7978	28.4647	62.7236	117.1496	198.5072	276.5994	360.3481	472.5779	617.2895
	NW	6.3701	16.0269	39.3719	79.4527	148.3375	224.5424	315.1169	453.8543	654.0191
	SW	6.9442	18.712	46.2433	93.6564	172.1031	254.7122	344.4808	498.4007	693.2603
SSGA	***CNT***	4.4106	11.3303	27.4308	51.231	88.6984	125.5644	163.5495	222.7366	295.1453
	JAC	2.9454	13.567	33.8169	58.8643	97.841	147.5644	202.3953	259.0705	332.9095
	GOW	2.9269	13.5336	34.0714	58.5213	98.9484	152.1767	190.1226	262.7954	333.9306
	SOK	3.0316	13.5033	32.4117	58.2222	99.5744	153.5518	191.6952	260.2008	332.153

	NLCS	5.044	24.0559	57.3629	112.2229	192.4885	282.085	370.822	511.8083	652.8444
	LEV	6.6624	28.7974	63.7158	117.5972	201.6532	282.7564	381.2036	518.3455	640.6613
	NW	6.5499	18.9662	50.1943	99.7192	181.1931	263.872	348.972	504.2265	662.3502

附录表 4-续 1

CSA		***NUCS***								
Search Algorithm	**Similarity Function**	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
SSGA	**SW**	6.003	22.3397	58.4595	114.549	202.1211	303.4799	389.775	532.7292	703.6118
RS	**CNT**	3.3798	13.8404	30.1692	56.8514	89.7823	131.0717	164.9947	213.916	299.1908
	JAC	3.8874	16.5271	36.753	66.7817	97.841	157.3546	200.1271	263.2746	336.8397
	GOW	3.8799	16.251	36.5485	66.681	103.9347	158.4033	199.9966	260.9527	338.5316
	SOK	3.9636	15.9329	35.7707	66.618	103.5387	157.3312	200.3379	259.7939	337.0229
	NLCS	6.8339	29.0003	64.9661	118.2967	181.7524	288.6232	372.5912	491.5114	582.0771
	LEV	7.3519	29.6209	65.4578	114.5871	179.5598	288.8658	378.2914	474.1072	587.4842
	NW	6.6767	27.7367	61.3369	114.3602	179.1779	272.6461	349.4743	471.85	573.0358
	SW	7.6562	31.9655	71.5579	129.8981	208.2383	313.6446	386.2711	538.1317	614.0757

参考文献

[1] Burns, A., Wellings, A.: Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX (4th Edition). Pearson Education Canada, (2009)

[2] RTCA: DO-178C Software Considerations in Airborne Systems and Equipment Certification. In. (2011)

[3] RTCA: DO-332 Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A. In. (2011)

[4] 邵维忠, 杨芙清: 面向对象的系统分析 (第 2 版). 清华大学出版社, (2006)

[5] Neill, C.J., Laplante, P.A.: Requirements Engineering: The State of the Practice. IEEE Software **6**, 40-45 (2003).

[6] Cox, K., Aurum, A., Jeffery, R.: An Experiment in Inspecting the Quality of Use Case Descriptions. Journal of Research and Practice in Information Technology **36**(4), 211-229 (2004).

[7] Yue, T., Briand, L.C., Labiche, Y.: A Use Case Modeling Approach to Facilitate the Transition towards Analysis Models: Concepts and Empirical Evaluation. In: MODELS 2009 2009, pp. 484-498. Springer (2009)

[8] Yue, T., Briand, L.C., Labiche, Y.: aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models. ACM Transactions on Software Engineering and Methodology (TOSEM) **24**(3), 13 (2015).

[9] Yue, T., Ali, S., Zhang, M.: RTCM: A Natural Language Based, Automated, and Practical Test Case Generation Framework. In: ISSTA 2015 2015, pp. 397-408. ACM (2015)

[10] Pohl, K.: Requirements Engineering: Fundamentals, Principles, and Techniques. Springer Publishing Company, Incorporated (2010)

[11] Fagan, M.E.: Advances in Software Inspections. IEEE Transactons on Software Engineering **12**(7), 744-751 (1986).

[12] Miller, J., Wood, M., Roper, M.: Further experiences with scenarios and checklists. Empirical Software Engineering **3**(1), 37-64 (1998).

[13] Bush, M.E., Fenton, N.E.: Software measurement: a conceptual framework. Journal of Systems and Software **12**(3), 223-231 (1990).

[14] Bush, M.: Formal Inspections–Do They Really Help? In: Proceedings of the Sixth Annual Conference of the National Security Industrial Association 1990

[15] Fagan, M.E.: Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal **15**(3), 182-211 (1976).

[16] Porter, A., Votta Jr, L.G., Basili, V.R.: Comparing detection methods for software requirements inspections: A replicated experiment. Software Engineering, IEEE Transactions on **21**(6), 563-575 (1995).

[17] Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M.V.: The Empirical Investigation of Perspective-Based Reading Empirical Software Engineering **1**(2), 133-164 (1996).

[18] Thelin, T., Runeson, P., Wohlin, C., Olsson, T., Andersson, C.: Evaluation of Usage-Based Reading—Conclusions after Three Experiments. Empirical Software Engineering **9**(1-

2), 77-110 (2004).
[19] Regnell, B., Runeson, P., Thelin, T.: Are the Perspectives Really Different?–Further Experimentation on Scenario-Based Reading of Requirements. Empirical Software Engineering **5**(4), 331-356 (2000).
[20] Alistair, C.: Writing effective use cases. Addison-Wesley, (2001)
[21] Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Professional, (2004)
[22] Guiney, E., Kulak, D.: Use Cases: Requirements in Context. Addison-Wesley, (2000)
[23] Yue, T., Briand, L.C., Labiche, Y.: Facilitating the Transition from Use Case Models to Analysis Models: Approach and Experiments. ACM Trans. TOSEM **22**(1), 5 (2013).
[24] Yue, T.: Automatically Deriving a UMLAanalysis Model from a Use Case Mmodel. Ph.D. Thesis, Carleton University, (2010)
[25] Cockburn, A.: Writing Effective Use Cases (1st Edition). Addison-Wesley Professional Reading (2000).
[26] Larman, C.: Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development. Pearson Education India, (2012)
[27] Bittner, K.: Use Case Modeling. Addison-Wesley Boston, (2002)
[28] Schneider, G., Winters, J.P.: Applying Use Cases: A Practical Guide (2nd Edition). Addison-Wesley Professional, (2001)
[29] OMG: Meta Object Facility (MOF) Core Specification V2.5. http://www.omg.org/spec/MOF/2.5/PDF/ (2015). Accessed Dec. 2, 2015
[30] Brown, E.K., Miller, J.: Syntax: a linguistic introduction to sentence structure. Psychology Press, (1991)
[31] Greenbaum, S.: The Oxford English Grammar, vol. 652. Oxford University Press Oxford, (1996)
[32] Van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Proceedings of Fifth IEEE International Symposium on Requirements Engineering 2001, pp. 249-262. IEEE (2001)
[33] Moreira, A., Chitchyan, R., Araújo, J., Rashid, A.: Aspect-Oriented Requirements Engineering. Springer, (2013)
[34] Grundy, J.: Aspect-Oriented Requirements Engineering for Component-Based Software Systems. In: Proceedings of IEEE International Symposium on Requirements Engineering 1999, pp. 84-91. IEEE (1999)
[35] Araujo, J., Whittle, J., Kim, D.-K.: Modeling and Composing Scenario-Based Requirements with Aspects. In: Proceedings of IEEE International Requirements Engineering Conference 2004, pp. 58-67. IEEE (2004)
[36] Brito, I.S., Moreira, A.: Towards an Integrated Approach for Aspectual Requirements. In: IEEE International Requirements Engineering Conference 2006, pp. 341-342. IEEE (2006)
[37] Zhang, J., Li, F., Zhang, Y.: Aspect-Oriented Requirements Modeling. In: 31st IEEE Software Engineering Workshop 2007, pp. 35-40. IEEE (2007)
[38] Zhang, J., Liu, Y., Jiang, M., Strassner, J.: An Aspect-Oriented Approach to Handling Crosscutting Concerns in Activity Modeling. In: Proceedings of the International MultiConference of Engineers and Computer Scientists 2008. IAENG (2008)

[39] Wehrmeister, M.A., Freitas, E.P., Pereira, C.E., Wagner, F.R.: An Aspect-Oriented Approach for Dealing with Non-functional Requirements in A Model-Driven Development of Distributed Embedded Real-Time Systems. In: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing 2007, pp. 428-432. IEEE (2007)

[40] Douglass, B.P.: Real Time UML: Advances in The UML for Real-Time Systems. Addison-Wesley Professional, (2004)

[41] 刘瑞成, 张立臣: 基于 UML 的面向方面的实时系统建模方法. 计算机应用 **25**(8), 1874-1877 (2005).

[42] 刘瑞成, 张立臣: 基于面向方面的实时系统建模方法. 计算机科学 **33**(7), 262-265 (2006).

[43] DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. Computer **11**(4), 34-41 (1978).

[44] Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. Software Engineering, IEEE Transactions on **37**(5), 649-678 (2011).

[45] Shahriar, H., Zulkernine, M.: Mutation-based testing of buffer overflow vulnerabilities. In: COMPSAC'08. 2008, pp. 979-984. IEEE

[46] Chevalley, P., Thévenod-Fosse, P.: A mutation analysis tool for Java programs. International journal on STTT **5**(1), 90-103 (2003).

[47] Offutt, A.J., Voas, J., Payne, J.: Mutation Operators for Ada. In. Technical Report ISSE-TR-96-09, Information and Software Systems Engineering, George Mason University, (1996)

[48] Fabbri, S.C., Delamaro, M.E., Maldonado, J.C., Masiero, P.C.: Mutation Analysis Testing for Finite State Machines. In: 5th International Symposium on Software Reliability Engineering, 1994, pp. 220-229. IEEE (1994)

[49] Mottu, J.-M., Baudry, B., Le Traon, Y.: Mutation analysis testing for model transformations. In: 2nd Conference ECMDA-FA., 2015 2006, pp. 376-390. Springer

[50] Sahin, D., Kessentini, M., Wimmer, M., Deb, K.: Model transformation testing: a bi-level search-based software engineering approach. Journal of Software: Evolution and Process **27**(11), 821-837 (2015).

[51] Schlick, R., Herzner, W., Jöbstl, E.: Fault-Based Generation of Test Cases from UML-Models - Approach and Some Experiences. In: International Conference on Computer Safety, Reliability, and Security 2011, pp. 270-283. Springer (2011)

[52] Di Nardo, D., Pastore, F., Briand, L.: Generating Complex and Faulty Test Data Through Model-Based Mutation Analysis. In: Software Testing, Verification and Validation (ICST) 2015, pp. 1-10. IEEE (2015)

[53] Mottu, J.-M., Baudry, B., Le Traon, Y.: Mutation Analysis Testing for Model Transformations. In: European Conference on Model Driven Architecture - Foundations and Applications 2006, pp. 376-390. Springer (2006)

[54] Offutt, J., Ammann, P., Liu, L.: Mutation Testing implements Grammar-Based Testing. In: Second Workshop on Mutation Analysis 2006. IEEE (2006)

[55] ISO: ISO/IEC 14977: 1996 (e), information technology syntactic metalanguage extended bnf. International Organization for Standardization (1996).

[56] Harman, M., Mansouri, S.A., Zhang, Y.: Search based software engineering: A

comprehensive analysis and review of trends techniques and applications. In. Technical Report TR-09-03, Department of Computer Science, King's College London, (2009)

[57] Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.M.: The next release problem. Information and Software Technology **43**(14), 883-890 (2001).

[58] Finkelstein, A., Harman, M., Mansouri, S.A., Ren, J., Zhang, Y.: A Search Based Approach to Fairness Analysis in Requirement Assignments to Aid Negotiation, Mediation and Decision Making. Requirements Engineering **14**(4), 231-245 (2009).

[59] Baker, P., Harman, M., Steinhöfel, K., Skaliotis, A.: Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In: 22nd IEEE International Conference on Software Maintenance, ICSM'06 2006, pp. 176-185. IEEE (2006)

[60] Zhang, Y., Harman, M., Mansouri, S.A.: The Multi-Objective Next Release Problem. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation 2007, pp. 1129-1137. ACM (2007)

[61] Saliu, M.O., Ruhe, G.: Bi-Objective Release Planning for Evolving Software Systems In: 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering 2007, pp. 105-114. ACM (2007)

[62] Haimes, Y.Y., Lasdon, L.S., Wismer, D.A.: On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization. IEEE Transactions on Systems, Man, and Cybernetics(1), 296-297 (1971).

[63] Srinivas, K., Gupta, M.: Software Requirements Selection using Quantum-inspired Elitist Multi-objective Evolutionary Algorithm. In: International Conference on Advances in Engineering, Science and Management (ICAESM) 2012, pp. 782-787. IEEE (2012)

[64] del Sagrado, J., del Águila, I.M., Orellana, F.J.: Multi-objective ant colony optimization for requirements selection. Empirical Software Engineering **20**(3), 577-610 (2015).

[65] Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. Information and Software Technology **46**(4), 243-253 (2004).

[66] Li, C., van den Akker, M., Brinkkemper, S., Diepen, G.: An integrated approach for requirement selection and scheduling in software release planning. Requirements Engineering **15**(4), 375-396 (2010).

[67] Carlshamre, P.: Release Planning in Market-Driven Software Product Development: Provoking an Understanding. Requirements Engineering **7**(3), 139-151 (2002).

[68] Karim, M.R., Ruhe, G.: Bi-objective Genetic Search for Release Planning in Support of Themes. In: Search-Based Software Engineering. pp. 123-137. Springer, (2014)

[69] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182-197 (2002).

[70] Li, Y., Yue, T., Ali, S., Zhang, L.: Zen-ReqOptimizer: A Search-Based Approach for Requirements Assignment Optimization. Empirical Software Engineering, 1-60 (2016).

[71] OMG: MARTE V1.1. http://www.omg.org/spec/MARTE/1.1/. Accessed Dec. 2, 2015

[72] IEEE: IEEE Recommended Practice for Software Requirements Specifications. IEEE Std. 830-1998 (1998).

[73] IEC: IEC 61882: 2001: Hazard and Operability Studies (HAZOP studies). Application

Guide. In: British Standards Institute. (2001)
[74] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media, (2012)
[75] Jedlitschka, A., Ciolkowski, M., Pfahl, D.: Reporting Experiments in Software Engineering. In: Guide to Advanced Empirical Software Engineering. pp. 201-228. Springer, (2008)
[76] Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J.: Preliminary Guidelines for Empirical Research in Software Engineering. IEEE Transactions on software engineering **28**(8), 721-734 (2002).
[77] Vargha, A., Delaney, H.D.: A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics **25**(2), 101-132 (2000).
[78] Kruskal, W.H., Wallis, W.A.: Use of Ranks in One-Criterion Variance Analysis. Journal of the American statistical Association **47**(260), 583-621 (1952).
[79] Wilcoxon, F., Katti, S., Wilcox, R.A.: Critical Values and Probability Levels for the Wilcoxon Rank Sum Test and the Wilcoxon Signed Rank Test. Lederle Laboratories, Division Amer. Cyanamid Company. Pearl River, New York. (1963).
[80] Wiegers, K.E.: Peer Reviews in Software: A Practical Guide. Journal of Object Technology **2**(1), 121-122 (2002).
[81] Zhang, M., Yue, T., Ali, S., Zhang, H., Wu, J.: A Systematic Approach to Automatically Derive Test Cases from Use Cases Specified in Restricted Natural Languages. In: SAM 2014 2014, pp. 142-157. Springer
[82] IBM: Rational Software Architect. https://www.ibm.com/developerworks/downloads/r/architect/.
[83] Zhang, G., Yue, T., Wu, J., Ali, S.: Zen-RUCM: A Tool for Supporting a Comprehensive and Extensible Use Case Modeling Framework. In: Demos/Posters/StudentResearch@ MoDELS 2013, pp. 41-45. Citeseer
[84] Alur, R., Courcoubetis, C., Dill, D.: Model-Checking for Real-Time Systems. In: LICS'90 1990, pp. 414-425. IEEE
[85] Alur, R., Henzinger, T.A.: Logics and Models of Real Time: A survey. In: Real-Time: Theory in Practice 1991, pp. 74-106. Springer
[86] Moser, L.E., Ramakrishna, Y., Kutty, G., Melliar-Smith, P.M., Dillon, L.K.: A graphical environment for the design of concurrent real-time systems. ACM Trans. TOSEM **6**(1), 31-79 (1997).
[87] Wegener, J., Mueller, F.: A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints. Real-Time Systems **21**(3), 241-268 (2001).
[88] Dromey, R.G.: From Requirements to Design: Formalizing the Dey Steps. In: SEFM 03 2003, pp. 2-11. IEEE
[89] Alur, R., Dill, D.: Automata For Modeling Real-Time Systems. In: Automata, languages and programming. Automata, Languages and Programming, pp. 322-335. Springer, (1990)
[90] Ramchandani, C.: Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. In. (1974)
[91] Holliday, M.A., Vernon, M.K.: A Generalized Timed Petri Net Model for Performance

Analysis. IEEE Trans. TSE(12), 1297-1310 (1987).
[92] OMG: SysML, V1.3. http://www.omg.org/spec/SysML/1.3/. Accessed Dec. 16, 2015
[93] OMG: SPT, V1.1. http://www.omg.org/spec/SPTP/1.1. Accessed Dec. 16, 2015
[94] OMG: UML, V2.4.1. http://www.omg.org/spec/UML/2.4.1. Accessed Dec. 16, 2015
[95] Wang, C., Pastore, F., Goknil, A., Briand, L., Iqbal, Z.: Automatic Generation of System Test Cases from Use Case Specifications. In: ISSTA 2015 2015, pp. 385-396. ACM
[96] Ali, S., Briand, L.C., Hemmati, H.: Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. Software & Systems Modeling **11**(4), 633-670 (2012).
[97] Sampaio, A., Rashid, A., Chitchyan, R., Rayson, P.: EA-Miner: towards automation in aspect-oriented requirements engineering. In: Transactions on aspect-oriented software development III. pp. 4-39. Springer, (2007)
[98] OMG: UML2.2. http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/ (2015). Accessed Dec. 16, 2015
[99] Sousa, G., Soares, S., Borba, P., Castro, J.: Separation of crosscutting concerns from requirements to design: Adapting the use case driven approach. In: Early Aspects 2004, pp. 93-102
[100] Chitchyan, R., Rashid, A., Rayson, P., Waters, R.: Semantics-based composition for aspect-oriented requirements engineering. In: Proceedings of the 6th international conference on Aspect-oriented software development 2007, pp. 36-48. ACM
[101] Behjati, R., Yue, T., Briand, L., Selic, B.: SimPL: A product-line modeling methodology for families of integrated control systems. Information and Software Technology **55**(3), 607-629 (2013).
[102] Zhang, H., Yue, T., Ali, S., Liu, C.: Facilitating Requirements Inspection with Search-Based Selection of Diverse Use Case Scenarios In: BICT 2015 2015, pp. 229-236. ICST (2015)
[103] Yue, T., Briand, L.C., Labiche, Y.: A systematic review of transformation approaches between user requirements and analysis models. Requirements Engineering **16**(2), 75-99 (2011).
[104] Somé, S.S.: Supporting use case based requirements engineering. Information and Software Technology **48**(1), 43-58 (2006).
[105] Anthonysamy, P., Somé, S.S.: Aspect-oriented use case modeling for software product lines. In: EA-AOSD'08 2008, p. 5. ACM
[106] Jacobson, I., Ng, P.-W.: Aspect-oriented software development with use cases (addison-wesley object technology series). Addison-Wesley Professional, (2004)
[107] Sillito, J., Dutchyn, C., Eisenberg, A.D., De Volder, K.: Use case level pointcuts. In: Odersky, M. (ed.) ECOOP 2004–Object-Oriented Programming, vol. LNCS., pp. 246-268. Springer, (2004)
[108] Mussbacher, G., Amyot, D., Weiss, M.: Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps. In: REV'06 2006. IEEE
[109] Alencar, F., Moreira, A., Castro, J., Silva, C., Mylopoulos, J.: Using Aspects to Simplify iModels. In: Requirements Engineering, 14th IEEE International Conference, Minneapolis/St. Paul, MN 2006, pp. 335-336. IEEE
[110] Jacobson, I.: Object-oriented software engineering: a use case driven approach. Pearson

Education India, (1993)
[111] OMG: Documents Associated With Business Process Model And Notation™ (BPMN™) Version 2.0. http://www.omg.org/spec/BPMN/2.0/PDF (2011). Accessed Dec. 2, 2016
[112] Kletz, T.A.: HAZOP and HAZAN: identifying and assessing process industry hazards. IChemE, (1999)
[113] Zhang, H., Wang, S., Yue, T., Ali, S., Wu, J., Liu, C.: Search and Similarity Based Selection of Use Case Scenarios: An Empirical Study. Empirical Software Engineering (2017, accepted).
[114] Offutt, A.J., Untch, R.H.: Mutation 2000: Uniting the Orthogonal. In: Wong, W.E. (ed.) Mutation testing for the new century. Mutation Testing for the New Century, pp. 34-44. Springer, (2001)
[115] Anda, B., Sjøberg, D.I.: Towards an inspection technique for use case models. In: 14th SEKE. 2002, pp. 127-134. ACM
[116] Denger, C., Paech, B., Freimut, B.: Achieving High Quality of Use-Case-Based Requirements. Informatik-Forschung und Entwicklung **20**(1), 11-23 (2005).
[117] Phalp, K.T., Vincent, J., Cox, K.: Assessing the quality of use case descriptions. Software Quality Journal **15**(1), 69-97 (2007).
[118] Yue, T., Ali, S.: Bridging the gap between requirements and aspect state machines to support non-functional testing: industrial case studies. In: ECMFA 2012. 2012. Modelling Foundations and Applications, pp. 133-145. Springer
[119] Wang, C., Pastore, F., Goknil, A., Briand, L., Iqbal, Z.: Automatic generation of system test cases from use case specifications. In: ISSTA 2015. 2015, pp. 385-396. ACM
[120] Just, R., Schweiggert, F.: Higher accuracy and lower run time: efficient mutation analysis using non‐redundant mutation operators. Software Testing, Verification and Reliability **25**(5-7), 490-507 (2015).
[121] Wu, J., Ali, S., Yue, T., Tian, J., Liu, C.: Assessing the quality of industrial avionics software: an extensive empirical evaluation. Empirical Software Engineering, 1-50 (2016).
[122] Gomaa, H.: Designing concurrent, distributed, and real-time applications with UML. Object Technology Series. Addison-Wesley, (2000)
[123] Capozucca, A., Cheng, B., Georg, G., Guelfi, N., Istoan, P., Mussbacher, G.: Requirements Definition Document For A Software Product Line Of Car Crash Management Systems. In. (2011)
[124] Pressman, R.S.: Software Engineering: A Practitioner's Approach (7th Edition). Palgrave Macmillan, (2010)
[125] Bruegge, B., Dutoit, A.H.: Object-Oriented Software Engineering Using UML, Patterns and Java (3rd Edition). Pearson, (2009)
[126] Gomaa, H.: Designing Concurrent, Distributed, and Real-Time Applications with UML. In: ICSE '01 Proceedings of the 23rd International Conference on Software Engineering 2001, pp. 737-738 ACM (2001)
[127] Capozucca, A., Cheng, B., Georg, G., Guelfi, N., Istoan, P., Mussbacher, G., Jensen, A., Jézéquel, J.-M., Kienzle, J., Klein, J.: Requirements Definition Document For A Software Product Line Of Car Crash Management Systems. ReMoDD repository, at http://www.cs.colostate.edu/remodd/v1/content/bcms-requirements-definition (2011).

[128] Basili, V.R., Caldiera, G., Rombach, H.D.: Goal question metric (gqm) approach. Encyclopedia of software engineering **1**, 528–532 (2002). doi:10.1002/0471028959.sof142

[129] Bonferroni, C.E.: Teoria statistica delle classi e calcolo delle probabilita. Libreria internazionale Seeber (1936)

[130] DeMillo, R., Lipton, R., Sayward, F.: Program mutation: A new approach to program testing. Infotech State of the Art Report, Software Testing **2**, 107-126 (1979).

[131] Arcuri, A., Briand, L.: A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In: 33rd International Conference on Software Engineering (ICSE) 2011, pp. 1-10. IEEE (2011)

[132] Kim, S., Clark, J., McDermid, J.: The Rigorous Generation of Java Mutation Operators Using HAZOP. In. Technical Report, 2/8/99, The University of York, (1999)

[133] Allenby, K., Kelly, T.: Deriving Safety Requirements Using Scenarios. In: 5th IEEE International Symposium on Requirements Engineering 2001, pp. 228-235. IEEE (2001)

[134] Seong, P.-H.: Reliability and Risk Issues in Large Scale Safety-critical Digital Control Systems. Springer Science & Business Media (2008)

[135] Porter, A., Votta Jr, L.G., Basili, V.R.: Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment IEEE Transactions on Software Engineering **21**(6), 563-575 (1995).

[136] Fusaro, P., Lanubile, F., Visaggio, G.: A Replicated Experiment to Assess Requirements Inspection Techniques. Empirical Software Engineering **2**(1), 39-57 (1997).

[137] Aurum, A., Petersson, H., Wohlin, C.: State-of-the-art: software inspections after 25 years. Software Testing, Verification and Reliability (STVR) **12**(3), 133-154 (2002).

[138] Wang, S., Ali, S., Yue, T., Li, Y., Liaaen, M.: A Practical Guide to Select Quality Indicators for Assessing Pareto-Based Search Algorithms in Search-Based Software Engineering. In: Proceedings of the 38th International Conference on Software Engineering 2016, pp. 631-642. ACM (2016)

[139] Cartaxo, E.G., Machado, P.D., Neto, F.G.O.: On the use of a similarity function for test case selection in the context of model-based testing. Software Testing, Verification and Reliability (STVR) **21**(2), 75-100 (2011).

[140] Xu, R., Wunsch, D.: Survey of Clustering Algorithms. IEEE Transactions on Neural Networks **16**(3), 645-678 (2005).

[141] Gusfield, D.: Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge university press, New York, NY, USA (1997)

[142] Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge university press, (1998)

[143] Simao, A.d.S., De Mello, R.F., Senger, L.J.: A Technique to Reduce the Test Case Suites for Regression Testing Based on a Self-Organizing Neural Network Architecture. In: 30th Annual International Conference on Computer Software and Applications (COMPSAC) 2006, pp. 93-96. IEEE (2006)

[144] Ledru, Y., Petrenko, A., Boroday, S.: Using String Distances for Test Case Prioritisation. In: Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on 2009, pp. 510-514. IEEE (2009)

[145] Hemmati, H., Arcuri, A., Briand, L.: Achieving Scalable Model-Based Testing Through

Test Case Diversity. ACM Transactions on Software Engineering and Methodology (TOSEM) **22**(1), 6 (2013).

[146] Laitenberger, O.: A Survey of Software Inspection Technologies. Handbook on Software Engineering and Knowledge Engineering **2**, 517-555 (2002).

[147] Ackerman, A.F., Buchwald, L.S., Lewski, F.H.: Software Inspections: An Effective Verification Process. IEEE Software **6**(3), 31-36 (1989).

[148] Humphrey, W.S.: A Discipline for Software Engineering. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (1995)

[149] Hemmati, H., Briand, L.: An Industrial Investigation of Similarity Measures for Model-Based Test Case Selection. In: IEEE 21st International Symposium on Software Reliability Engineering (ISSRE) 2010, pp. 141-150. IEEE (2010)

[150] Dong, G., Pei, J.: Sequence Data Mining, vol. 33. Springer Science & Business Media, (2007)

[151] Harman, M.: The Current State and Future of Search Based Software Engineering. In: 2007 Future of Software Engineering 2007, pp. 342-357. IEEE Computer Society (2007)

[152] Korel, B.: Automated Software Test Data Generation. IEEE Transactions on Software Engineering **16**(8), 870-879 (1990).

[153] Harman, M., McMinn, P.: A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search. IEEE Transactions on Software Engineering **36**(2), 226-247 (2010).

[154] Golberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, (1989)

[155] Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press Cambridge, MA, USA (1992)

[156] Deb, K., Kumar, A.: Real-coded Genetic Algorithms with Simulated Binary Crossover: Studies on Multimodal and Multiobjective Problems. Complex Systems **9**(6), 431-454 (1995).

[157] Haupt, R.L., Haupt, S.E.: Practical Genetic Algorithms, 2nd Edition. John Wiley & Sons, Inc., Hoboken, New Jersey (2004)

[158] Vavak, F., Fogarty, T.C.: Comparison of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments. In: Proceedings of IEEE International Conference on Evolutionary Computation 1996, pp. 192-195. IEEE (1996)

[159] Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction, vol. 16. John Wiley & Sons, (2001)

[160] Whitley, L.D.: The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In: Proceedings of the third international conference on Genetic algorithms 1989, pp. 116-123 (1989)

[161] Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276**(1), 51-81 (2002).

[162] Ali, S., Briand, L.C., Hemmati, H., Panesar-Walawege, R.K.: A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation. IEEE Transactions on Software Engineering **36**(6), 742-762 (2010).

[163] Harman, M., Jones, B.F.: Search-based software engineering. Information and Software

Technology **43**(14), 833-839 (2001).
[164] Arcuri, A.: It really does matter how you normalize the branch distance in search-based software testing. Software Testing, Verification and Reliability (STVR) **23**(2), 119-147 (2013).
[165] Ali, S., Iqbal, M.Z., Arcuri, A., Briand, L.C.: Generating Test Data from OCL Constraints with Search Techniques. IEEE Transactions on Software Engineering **39**(10), 1376-1402 (2013).
[166] Hajri, I., Goknil, A., Briand, L.C., Stephany, T.: Applying Product Line Use Case Modeling in an Industrial Automotive Embedded System: Lessons Learned and a Refined Approach. In: ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS) 2015, pp. 338-347. IEEE (2015)
[167] Kendall, M.G.: A New Measure of Rank Correlation. Biometrika **30**, 81-93 (1938). doi:10.2307/2332226
[168] Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures (5th edition). Chapman and Hall/CRC (2011)
[169] Durillo, J.J., Nebro, A.J.: jMetal: A Java framework for multi-objective optimization. Advances in Engineering Software **42**(10), 760-771 (2011).
[170] Yue, T., Ali, S.: Applying Search Algorithms for Optimizing Stakeholders Familiarity and Balancing Workload in Requirements Assignment. In: Proceedings of the 2014 conference on Genetic and evolutionary computation 2014, pp. 1295-1302. ACM (2014)
[171] Arcuri, A., Fraser, G.: On parameter tuning in search based software engineering. In: Search Based Software Engineering. pp. 33-47. Springer, (2011)
[172] Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology **48**(3), 443-453 (1970).
[173] Gilb, T., Graham, D., Finzi, S.: Software Inspection. Addison-Wesley Longman Publishing Co., Inc., (1993)
[174] Martin, J., Tsai, W.T.: N-Fold Inspection: A Requirements Analysis Technique Communications of the ACM **33**(2), 225-232 (1990).
[175] Kollanus, S., Koskinen, J.: Survey of Software Inspection Research. The Open Software Engineering Journal **3**(1), 15-34 (2009).
[176] Anda, B., Sjøberg, D.I.: Towards an Inspection Technique for Use Case Models. In: Proceedings of the 14th international conference on Software engineering and knowledge engineering 2002, pp. 127-134. ACM (2002)
[177] Zhang, H., Yue, T., Shaukat, A., Liu, C.: Towards Mutation Analysis for Use Cases. In: ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS'16), October 2-7 2016, pp. 363-373. ACM (2016)
[178] Bernárdez, B., Genero, M., Durán, A., Toro, M.: A Controlled Experiment for Evaluating a Metric-Based Reading Technique for Requirements Inspection. In: 10th International Symposium on Software Metrics 2004, pp. 257-268. IEEE (2004)
[179] Naveed, A., Ikram, N.: A Novel Checklist: Comparison of CBR and PBR to Inspect Use Case Specification. In: Requirements Engineering in the Big Data Era. pp. 109-125. Springer, (2015)
[180] Sinha, A., Sutton, S.M., Paradkar, A.: Text2Test: Automated Inspection of Natural

Language Use Cases. In: 3rd International Conference on Software Testing, Verification and Validation (ICST) 2010, pp. 155-164. IEEE (2010)

[181] Fantechi, A., Gnesi, S., Lami, G., Maccari, A.: Applications of linguistic techniques for use case analysis. Requirements Engineering **8**(3), 161-170 (2003).

[182] Cox1, K., Aurum, A., Jeffery, R.: A Use Case Description Inspection Experiment. In. UNSW-CSE-TR-0414, University of New South Wales, School of Computer Science and Engineering., (2004)

[183] Durán, A., Ruiz-Cortés, A., Corchuelo, R., Toro, M.: Supporting Requirements Verification Using XSLT. In: IEEE Joint International Conference on Requirements Engineering 2002, pp. 165-172. IEEE (2002)

[184] Jacobson, I., Spence, I., Bittner, K.: USE-CASE 2.0 The Guide to Succedding with Use Cases. (2011).

攻读博士学位期间所取得的研究成果

主要学术论文

[1] **Huihui Zhang**, Tao Yue, Shaukat Ali and Chao Liu. Search and Similarity Based Selection of Use Case Scenarios: An Empirical Study [J]. Empirical Software Engineering (**EMSE**), online published, pp. 1-78, (doi: 10.1007/s10664-017-9500-x), 29 April 2017. (SCI, IF_{2015}: 1.393, CCF-B 期刊).

[2] **Huihui Zhang**, Tao Yue, Shaukat Ali and Chao Liu. Towards Mutation Analysis for Use Cases [C]. In proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (**MODELS**), pp. 363-373, ACM, 2016. (CCF-B 会议, EI, MODELS 2016 最佳论文提名)

[3] **Huihui Zhang**, Tao Yue, Shaukat Ali, Ji Wu and Chao Liu. A Restricted Natural Language Based Use Case Modeling Methodology for Real-Time Systems [C]. In proceedings of the 9th Workshop on Modelling in Software Engineering (MiSE) of the 2017 IEEE/ACM 39th International Conference on Software Engineering (**ICSE**) 2017. (CCF-A 会议, EI, 已录用).

[4] **Huihui Zhang**, Tao Yue, Shaukat Ali and Chao Liu. Facilitating requirements inspection with search-based selection of diverse use case scenarios [C]. In proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (**BICT** formerly BIONETICS), pp. 229-236. ACM, 2016. (EI)

[5] Tao Yue, **Huihui Zhang**, Shaukat Ali and Chao Liu. A Practical Use Case Modeling Approach to Specify Crosscutting Concerns [C]. In proceedings of International Conference on Software Reuse (**ICSR**), pp. 89-105. Springer, 2016. (CCF-C 会议, EI)

[6] Ji Wu, Tao Yue, Shaukat Ali and **Huihui Zhang**. A Modeling Methodology to Facilitate Safety-Oriented Architecture Design of Industrial Avionics Software [J]. Software: Practice and Experience (SPE), 5 (7), 893-924, DOI: 10.1002/spe.2281, WILEY, July 2015. (SCI, IF_{2015}= 0.652, CCF-B 期刊)

[7] Ji Wu, Tao Yue, Shaukat Ali and **Huihui Zhang**. Ensuring Safety of Avionics Software at the Architecture Design Level: An Industrial Case Study [C]. In proceedings of the

13th International Conference of Quality Software (QSIC), pp.55 – 64. IEEE, 2013. (CCF-C 会议, EI)

[8] Man Zhang, Tao Yue, Shaukat. Ali, **Huihui Zhang** and Ji Wu. A Systematic Approach to Automatically Derive Test Cases from Use Cases Specified in Restricted Natural Languages [C]. In proceedings of the 8th System Analysis and Modelling Conference (SAM'14), pp. 142-157. Springer, 2014. (EI)

授权的发明专利

[1] 吴际, 张辉辉, 李亚晖, 牛文生. 一种适用于航电系统的安全需求建模方法. 专利状态: 授权, 专利号: CN201310595322.0.

致　谢

在北航软件所度过的七年博士生活是我人生旅途中一个难忘的篇章，它开启了我职业研究的航程。在漫长的求学过程中，我的家人、身边的良师益友给予我无私的帮助，对他们致以最诚挚的感谢！

感谢我的奶奶、我的父母、我的妻子，是你们的支持和无私奉献使我能够坚持这场持久战。

感谢校内导师刘超教授，在漫长的七年博士生活中，刘老师的科研态度和工作热情总使我勇于面对各种困难。感谢校外导师 Dr. Tao Yue (Chief Research Scientist, Simula Research Laboratory) 和 Dr. Shaukat Ali (Senior Research Scientist, Simula Research Laboratory)，你们的职业修养、科研敏锐性和前瞻性、严谨的科研作风，总能激励我不断实现自我突破。

感谢课题组的吴际老师、杨海燕老师、任健老师，你们的辛勤付出使得课题组有条不紊的运作，为同学们提供优良的科研环境。

北航的博士生活给予我基本的科研素养，它将开启我职业研究的新航程。

作者简介

张辉辉，男，1981 年 9 月生于山东省安丘市。2010 年 6 月于北京大学获得软件工程硕士学位。2006 年 6 月于曲阜师范大学获得计算机科学与技术学士学位。2015 年 7 月 1 日-2016 年 8 月 1 日，参加国家公派留学赴挪威 Simula Research Laboratory 进行博士课题研究。主要研究兴趣：需求工程(RE)、RUCM 用况建模、实证软件工程(EMSE)、基于搜索的软件工程(SBSE)、模型驱动工程(MDE)。