软件集成开发环境的技术债务管理研究

刘亚珺[1] 李 兵[1,2] 李增扬[1,2] 梁 鹏[3] 吴闽泉[3]
(武汉大学国际软件学院 武汉 430072)[1] (武汉大学复杂网络研究中心 武汉 430072)[2]
(武汉大学计算机学院软件工程国家重点实验室 武汉 430072)[3]

摘 要 软件技术债务是运用经济学中"债务"的概念来描述软件开发中因项目短期利益而实施的技术折中。但从长期来看,技术债务会影响软件的质量、成本和开发效率,因此有必要对其进行有效管理。现有的技术债务管理工具数量少且存在各种局限性,难以实现有效的管理。主流的软件集成开发环境功能强大且应用广泛,可以为技术债务管理服务。以具有代表性的集成开发环境Visual Studio 2015企业版为研究对象,通过C#实例发现其管理4类与代码直接相关的技术债务的能力,并将其与4种专门的技术债务管理工具进行对比,为开发团队的日常实践提供技术债务管理支持。结果表明,Visual Studio能够提供更好的技术债务管理功能,并能应用多种方法对项目中存在的各类技术债务进行不同程度的管理。

关键词 技术债务,技术债务类型,技术债务管理活动,技术债务管理工具,Visual Studio
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.003

Study on Technical Debt Management of Integrated Development Environment

LIU Ya-jun[1] LI Bing[1,2] LI Zeng-yang[1,2] LIANG Peng[3] WU Min-quan[3]
(International School of Software, Wuhan University, Wuhan 430072, China)[1]
(Research Center of Complex Network, Wuhan University, Wuhan 430072, China)[2]
(State Key Laboratory of Software Engineering, School of Computer Science, Wuhan University, Wuhan 430072, China)[3]

Abstract Technical debt (TD), a metaphor to financial debt, refers to technical compromises which are made to gain short-term benefits at the cost of long-term software quality. However, in the long run TD will negatively affect software quality as well as cost and productivity of software development. Thus, TD should be effectively managed. Existing dedicated TD management tools are rather limited and have various limitations. Hence, it is difficult to effectively manage TD using such tools. Mainstream integrated development environments (IDEs) are powerful and widely used, can serve to manage TD. In this study, we investigated Microsoft Visual Studio 2015 Enterprise Edition, a representative mainstream IDE, to explore its ability for managing four types of TD directly related to source code using a C# project. Then we compared Visual Studio with four dedicated TD management tools with respect to their capabilities in managing TD and their support for TD management of development teams in their daily practices. The results show that Visual Studio provides better support for managing TD in terms of diversity of TD types and the range of TD management activities. In addition, with Visual Studio, specific TD instances can be managed to different extents in a variety of ways.

Keywords Technical debt, Technical debt type, Technical debt management activity, Technical debt management tool, Visual Studio

1 引言

技术债务(Technical Debt, TD)是一种反映技术折中的隐喻说法,为了短期的利益而采取欠佳的技术决策,可能会对软件系统的长期质量产生不良影响。该隐喻最早主要关注软件实现,即软件的代码层,现已逐渐扩展到软件架构、详细设计以及文档、需求和测试等方面[1]。早在1992年,Ward Cunningham就在OOPSLA报告[2]中提出了技术债务的概念并指出了其对软件开发的影响,但直至近几年,技术债务才受到工业界和学术界的重点关注。

到稿日期:2016-10-21 返修日期:2016-12-22 本文受国家重点研发计划(2016YFB0800405),国家重点基础研究发展计划(2014CB340404),国家自然科学基金(61572371,61472286),中国博士后基金(2015M582272),中央高校基本科研业务费专项资金(2042016kf0033),湖北省自然科学基金(2016CFB158)资助。

刘亚珺(1992-),女,硕士生,主要研究方向为技术债务与软件架构,E-mail:liuyajun@whu.edu.cn;李 兵(1969-),男,博士,教授,主要研究方向为网络化软件、面向服务的软件工程、复杂系统与复杂网络、人工智能、云计算与大数据;李增扬(1982-),男,博士后,讲师,主要研究方向为技术债务、软件架构、复杂系统与复杂网络;梁 鹏(1978-),男,博士,教授,主要研究方向为软件体系结构、需求工程;吴闽泉(1966-),男,硕士,高级工程师,主要研究方向为计算机应用,E-mail:rjgc@whu.edu.cn(通信作者)。

在对技术债务的描述中，同样使用了金融债务中的一些概念来进行类比，如本金、利息、效益、风险等。本金即消除技术债务的成本；利息是在不解决技术债务的情况下对软件系统进行维护与扩展所需的额外成本；效益指引入技术债务后可获得的短期收益；技术债务也是软件项目风险的一类，因为如果技术债务持续累积，将影响系统的质量[3]。

技术债务的产生有多种原因：可能是项目利益相关者为了获得短期利益而有意引入的，比如项目团队为了尽可能获得更高的商业价值，加快产品新特性的开发，尽早将产品投入市场[2]，以帮助公司在竞争中取得领先；或者是由于产品开发受到预算的限制等，只能相应地做出技术上的妥协。尽管在未来可能需要付出比当前更高的代价来解决这些技术债务，但如果这种有意引入的技术债务的成本、利息及对系统的影响是已知的、可控的，这样的妥协是可以接受的[4]。另一方面，技术债务的引入可能是无意的，项目经理和开发团队没有意识到它在项目中的存在和影响[5]，如使用了旧的技术、工具或者标准；开发人员缺乏一定的专业知识[6]。如果技术债务是未知的，且未得到解决，将会不断累积，进而给软件系统的后期维护和演化带来巨大挑战。

为了改善软件系统的可维护性、可重用性、可理解性，同时防止累积的技术债务对软件质量造成难以逆转的损害，无论是有意还是无意产生的技术债务，对其进行相应的控制和管理是非常必要的[7]。试想如果在项目的某个组件中引入了技术债务，之后又基于这个组件对项目不断地进行扩展，若不能及时且有效地对技术债务进行纠正，这样不断累积的技术债务将会使软件的维护和演化变得越来越困难，从而增加更多的技术债务，这种恶性循环在极端情况下可能会导致软件产品的“破产”和项目的终止。

对技术债务的管理主要包括两个方面：1）预防潜在技术债务（有意的和无意的）的产生；2）对现有技术债务的处理（识别、偿还等），使其变得可见和可控，防止其在软件项目中的过度累积，同时平衡软件项目的成本和价值[8]。为了能有效地管理技术债务，开发团队必须对技术债务的类型（如代码TD、设计 TD、架构 TD、测试 TD 等）、产生原因以及影响等方面有足够的认识，并了解技术债务管理的现状，包括已被提出、开发、使用的技术债务管理方法和工具等。

软件技术债务作为一个较新的研究领域，在工业界和学术界的相关研究和应用正逐步兴起，但能用于技术债务管理的工具十分有限。根据我们之前所做的技术债务管理方面的映射研究（Systematic Mapping Study）[5]发现，现有的技术债务管理工具有 4 个方面的局限性：1）仅针对一种或两种技术债务实例，如 Software maps tool[9]和 SonarQube[10]等工具仅适用于管理代码技术债务；2）仅支持特定技术债务管理活动，大部分工具适用于识别技术债务，如 SonarQube 和 NDepend[1]，少数可用于度量技术债务，如 Software maps tool；3）支持的编程语言单一，如 DebtFlag[11] 仅支持 Java 语言；4）工具需额外部署使用，不能集成到主流开发环境中，不利于开发人员的日常使用。

考虑到与代码相关的技术债务（如代码 TD、设计 TD）会随着项目代码的演化而频繁地变更，技术债务管理工具应尽可能促进开发团队的日常开发活动。现有集成开发环境没有提供专门管理技术债务的功能，但集成开发环境的某些功能可以为管理技术债务提供支持。因此，探究目前使用广泛、功能强大的集成开发环境是否同样具有技术债务管理能力对于项目开发十分重要，有利于开发者在开发功能的同时时刻关注并提升代码质量。Microsoft Visual Studio（VS）作为目前最流行的集成开发环境之一，被大量开发团队用于实现诸多类型和不同规模的软件项目，能够支持 C＃，C＋＋，JavaScript，F＃，Visual Basic，Python 等多种编程语言，有较强的实用性。不同版本的 VS 产品中，VS 2015 企业版功能最为强大，适用于具有高质量要求和扩展需求的各种规模的开发团队，在测试（代码覆盖率、自动化单元测试等）、体系结构与建模、代码分析等方面的功能完备性是其他版本不能比拟的。因此，本文选择 VS 2015 企业版作为开发环境，力求充分探究其作为主流集成开发环境对技术债务管理的支持程度。

本文首先介绍技术债务按照软件生命周期的不同阶段进行的分类及每种技术债务的成因，并概述技术债务管理过程所包含的各项管理活动（如识别、度量、偿还）；然后通过一个C＃项目实例，具体分析 VS 可管理的与代码直接相关的技术债务类型及其实例，以及支持的管理活动及具体实现方法，并将 VS 与 4 种专门的技术债务管理工具进行比较分析。结果表明，尽管 VS 不能对项目中具体技术债务实例实现完整的管理过程，但是相比已有的技术债务管理工具，VS 能够采取多种方法对项目中与代码直接相关的技术债务进行不同程度的有效管理，从而更好地改善软件质量。

本文第 2 节结合我们之前所做的技术债务映射研究[5]，介绍按照软件开发生命周期中不同阶段划分的技术债务类型，概述技术债务管理过程中的各项活动；第 3 节依据本文的研究目的，提出研究问题，设计研究过程；第 4 节展示 VS 管理技术债务能力的研究结果，回答研究问题；第 5 节讨论技术债务管理对开发团队的指导意义，指出 VS 在技术债务管理方面的优点与不足，以及本文研究的局限性；最后总结全文并指出未来的工作方向。

2 技术债务管理

本节主要根据软件生命周期的不同阶段和技术债务的成因，对目前所了解的技术债务进行详细的分类并对分类结果进行呈现，同时介绍技术债务管理活动和相应的管理方法。

2.1 技术债务的类型

对技术债务进行有效的管理，首先要明确技术债务的具体类型以及成因，了解在软件开发过程中不良的技术决策可能会引入的技术债务，从而在软件项目中识别技术债务的存在，并对其进行管理。

技术债务的类型是指技术债务的一个特定分类（如架构、设计、代码、测试）以及进一步基于技术债务成因的一个子类（如代码方面的技术债务可能是由重复的代码导致的）。在我

1) http://www.ndepend.com/

们之前所做的映射研究[5]中，根据软件开发生命周期的不同阶段将具体技术债务分为 10 类，并依据技术债务的成因列举了几个子类。

(1)需求 TD：在领域假设和约束下，最优需求规范和实际系统实现之间的差距。例如，过度需求使系统变得复杂冗余，同时缺陷产生的可能性增大，不仅降低了系统的可维护性，而且增加了实现成本。

(2)架构 TD：由架构决策导致的，在诸如可维护性等内部质量方面做出的妥协。架构 TD 包括架构模块之间不良的依赖关系，以及一些典型的架构坏味(Architecture Smells)，如连接器嫉妒(Connector Envy)[12]，即组件覆盖了太多本应委托给连接器的交互相关功能(通信、协调、转换、促进优化)，这种耦合了连接器功能的组件会降低系统的重用性、可理解性和可测试性。

(3)设计 TD：在详细设计中采取的一些技术妥协，如复杂的类或方法。这些设计 TD 会降低系统的可理解性、可重用性[13]。

(4)代码 TD：编写欠佳的代码，违反了编码最佳实践或编码规则。比如，代码重复会加大软件系统更改的难度，因为必须找到并更新多个代码片段。代码 TD 是技术债务相关文献中最普遍提及的类型，大部分工具也被用于管理代码 TD。

(5)测试 TD：测试中采取的一些捷径。比如，缺少测试(单元测试、集成测试等)；低代码覆盖率，即测试代码对被测试功能代码分支的覆盖程度低。

(6)构建 TD：软件构建系统中存在的，或是使软件构建过程过于复杂和困难的缺陷。如不良的构建依赖，一方面，构建并维护系统中不必要的依赖关系会降低系统整体的构建和测试速度；另一方面，低层文件库在移除高层文件库中未声明的依赖时会丢失一些必需的依赖关系，进而导致构建的破损[14]。

(7)文档 TD：软件开发中任何不充分、不完整或者过期的文档，如过期的架构文档，缺少代码注释等。

(8)基础设施 TD：与开发相关的过程、技术、支持工具等方面的次优配置。次优的配置(如使用旧技术)会降低团队开发高质量产品的能力。

(9)版本控制 TD：源代码版本控制的问题，如不必要的代码分支[15]。

(10)缺陷 TD：软件系统中发现的缺陷、漏洞或者故障。这一技术债务类型是根据已有文献归纳所得，但对这一类型的技术债务存在争议。我们认为，缺陷、漏洞和故障会直接影响到最终用户的使用体验，不属于技术债务的范畴。

明确的技术债务分类不仅有利于对技术债务进行有针对性的管理，而且有助于不同角色的利益相关者(如需求工程师、架构师、编程人员、测试工程师)关注他们所参与的软件开发阶段中可能产生的技术债务。

2.2　技术债务管理活动

如不对技术债务加以管理，将会对软件系统的开发和维护造成不利的影响，包括降低软件产品的质量，增加后期维护及更新的成本，降低开发团队的工作效率等。因此，管理控制软件系统中的技术债务是非常必要的。技术债务管理由一系列可预防潜在技术债务的发生以及对现有技术债务进行处理以使其保持在合理程度的活动组成。下面将详细介绍技术债务管理活动，并且针对不同的管理活动列举一些可用的方法，为软件项目的开发者管理技术债务提供指导和支持。

(1)TD 识别：通过特定技术，检测软件系统中由欠佳的技术决策所导致的技术债务[16]。例如，通过源代码分析方法识别出违反编码规则的部分；基于源代码计算软件度量值，识别设计问题；通过分析不同类型软件元素(组件、模块等)之间的依赖性，可识别诸如循环依赖等技术债务。

(2)TD 度量：利用估算方法来量化软件系统中已知技术债务的效益和成本，或者估量系统的整体技术债务水平。通过源代码的度量值、数学方程和模型计算技术债务；依照经验和专业知识，人为估计[17]技术债务。

(3)TD 表示/文档化：以统一的方式表示和记录技术债务——技术债务项，进而引起特定利益相关者的关注。“技术债务项”是软件系统中的技术债务单元，通常包含唯一 ID 标识、位置信息、偿还负责人、技术债务类型以及描述信息。

(4)TD 优先级排序：依照某些预定义的规则对已识别的技术债务项排序，以此来决定应优先偿还的技术债务项或可暂时搁置的技术债务项。技术债务优先级排序可使用成本/效益分析方法，优先偿还具有更高的偿还成本/收益比的技术债务项，即如果解决一个技术债务项可产生更高的效益，则偿还该技术债务项[18]。

(5)TD 预防：目的是预防潜在技术债务的产生。可通过改善当前的开发过程来预防技术债务的引入；或使用架构决策支持方法，即估计不同架构设计决策可能导致的潜在技术债务，然后选择有较少潜在技术债务的决策[19]。

(6)TD 监测：观察系统中尚未解决的技术债务以及它的成本和效益随时间的变化情况。可采用基于阈值的方法为技术债务的相关质量度量值设置阈值，如未达到阈值则发出警告[20]；或根据系统所包含的技术债务以及组件之间的依赖关系，跟踪技术债务的传播[11]。

(7)TD 偿还：通过再工程或重构等方法，彻底解决或减轻软件系统中的技术债务。重构是最常见的技术债务偿还方法，即改善软件系统的内部质量，对软件的代码、设计或架构做出相应的改变，以适应未来软件维护和发展的需要，但是重构并不改变软件系统的外部行为；自动化也是一种技术债务偿还方法，通过将需手动重复的工作(如手动测试、手动部署)自动化，进而改进系统的可维护性，提高了开发效率和质量。

(8)TD 交流：使已识别的技术债务对利益相关者可见，这样有利于开发团队的讨论和管理。如可视化代码度量值或依赖关系，高亮显示组件中技术债务的位置。

实际上，技术债务管理的目标并不是项目零债务。好的技术债务管理是对商业目标和软件质量的平衡，若一味追求质量上的完美，可能会对软件开发的效率、可持续性，甚至战略目标造成严重的阻碍。在有限的资源条件以及竞争激烈的市场环境下，系统存在技术债务是不可避免的。从商业的视角看来，减少项目的技术债务、改善软件的内部质量，只有在其能增加收益的情况下才是有效的。然而，如果所累积的技术债务阻碍了软件新功能的增加，可能会迫使公司暂时停止增加项目的商业价值，转而投入全部的精力来维护系统的运

营。简而言之,如果不对技术债务加以管理,企业在该项目上可能会停止获利[21]。

3 研究设计

本节主要阐述本文研究的目的,并依据研究目的提出研究问题,设计对 Visual Studio 管理技术债务能力的研究过程。

3.1 研究目的和研究问题

本文的研究目的是更好地理解技术债务管理的相关理论与实践,以便在实际软件开发中利用主流集成开发环境 VS 对技术债务的管理能力,提高软件的质量和可维护性。这种能力主要体现在两方面:它能够管理的技术债务类型和具体实例的广泛性,以及它对不同技术债务类型的管理程度,包括可采取的方法和技术。基于该研究目标,定义以下研究问题(Research Question,RQ)。

RQ1:Visual Studio 支持对哪些技术债务类型的管理?

通过此研究问题,我们希望明确 VS 具备管理哪些类型的技术债务的能力;并进一步明确对于不同的技术债务类型,VS 可具体管理的技术债务实例。

RQ2:Visual Studio 支持哪些技术债务管理活动?

通过此研究问题,我们希望理解 VS 对技术债务管理的支持力度,以及 VS 实现每个技术债务管理活动的具体操作。

RQ3:Visual Studio 与专门的技术债务管理工具在技术债务管理能力上有何不同?

通过此研究问题,我们希望了解相比专门的技术债务管理工具,VS 在能够管理的技术债务类型和支持的技术债务管理活动方面所体现的优势和不足。

3.2 研究过程

对 VS 的实践研究所使用的案例是作者实验室一个现有的 C# 项目。之所以选择编程语言为 C# 的项目,是考虑到 VS 的核心是基于.NET 框架开发的,而 C# 是由微软所开发的编写.NET 框架的程序设计语言。对于 C# 项目,VS 所能提供的功能将会更加全面。

我们之前所做的映射研究[5]的结果显示:大部分相关文献所研究的技术债务类型集中于代码、设计和架构 3 个方面;同时,由于 VS 集成开发环境主要适用于对代码的开发和管理,并且提供了对项目较为完整的测试体系,因此本文主要针对代码 TD、设计 TD、架构 TD 以及测试 TD 这 4 种类型来探究 Visual Studio 的技术债务管理能力。为了实现上述目标,将依照以下步骤对 VS 进行案例实践研究:首先,在 VS 中打开该 C# 项目,并将其源代码作为输入;然后,按从左至右、从上至下的顺序了解 VS 的每个菜单功能项,重点关注能帮助改善代码质量以及开发过程的相关功能;接着,在实际操作之前,先绘制一个二维的技术债务类型及其管理活动的对应表,以便记录考查结果;最后,对 C# 项目依次运行每个功能项,并对照已有的技术债务类型和管理活动来判断该功能是否适用于技术债务及其管理,如果适用,则在表中类型和活动相对应的位置详细地记录下运行结果、技术债务实例、使用管理方法或技术、VS 的功能操作。

我们之前所做的技术债务映射研究[5]中总结了研究文献中所提出、使用或开发的可实现技术债务管理目的的工具。本文选择其中专门用于技术债务管理的 4 种工具(Software maps tool[22], Technical Debt Evaluation plugin for SonarQube[23], DebtFlag[24], Sonar TD plugin[25]),分别探究它们能够管理且与代码直接相关的技术债务类型以及采取的相应管理活动,将结果汇总并与 VS 的管理能力进行比较。

在第 4 节的研究结果中,将根据这些实验结果以及汇总数据具体分析并回答在 3.1 节中所提出的与 VS 管理技术债务能力相关的研究问题。

4 研究结果

依照 3.2 节所描述的实验过程对 Visual Studio 管理技术债务的能力进行研究。4.1 节和 4.2 节分别展示 VS 支持的技术债务类型和管理活动,对应回答了 3.1 节提出的 RQ1 和 RQ2;4.3 节展示了 VS 与 4 种技术债务管理工具对比汇总后的研究结果,回答了 RQ3。

4.1 VS 可管理的技术债务类型(RQ1)

相比大部分工具是针对某种特定类型的技术债务,本文探究到 VS 集成开发环境可以管理的具体技术债务种类较为广泛,尤其对于项目中架构、设计、代码、测试等方面的技术债务。表 1 列出了 VS 所涉及的具体技术债务种类和实例。

表 1 技术债务类型和实例

技术债务类型	技术债务实例	描述	Visual Studio 实例功能
架构 TD	不良的架构依赖	项目、模块级别的循环引用、依赖项过多等	代码图分析器
	代码与预期架构冲突	代码与预期设计的系统架构之间的不一致	层关系图验证
设计 TD	设计冗余	不必要的调用或声明,如不必要的“using”、未使用过的变量	运行代码分析、组织 using
	复杂的类或方法	负责执行的功能过多	代码度量值(代码行数)
	不良的设计依赖	类之间的循环引用、过于耦合等	代码图分析器、代码度量值(继承深度、类耦合度)
代码 TD	重复代码	完全相同或十分相似的代码段	分析代码克隆
	低质量代码	代码质量差、不利于维护	代码度量值(可维护性指数)
	复杂的代码	在类或方法内部代码逻辑结构过于复杂,难以理解	代码度量值(圈复杂度)
	编码规则冲突	不符合编码规范的、影响可维护性的代码段	运行代码分析
测试 TD	低代码覆盖率	代码覆盖率指编码的测试占实际进行测试的项目代码的比例	分析代码覆盖率
	缺少测试	缺少用于检验代码正确性和功能完整性的测试	运行项目测试

下面对几种关键且难以理解的技术债务实例给予说明。

(1)不良的依赖关系

VS 中可以通过创建代码图将代码元素(如类、程序集)之间的依赖关系可视化,帮助开发者理解代码结构,同时可以发现结构中不良的依赖关系,这对于大型复杂项目的开发和维护尤其重要。通过代码图不但可以查看整体依赖关系,还可

以对指定元素查看更深层次的代码依赖关系，直到类和成员级别。

利用 VS 提供的代码图分析器，能识别出代码中可能存在的循环依赖、依赖项过多、无依赖项这 3 种潜在的技术债务。出现循环依赖或依赖项过多的代码段不利于更改、测试和重用，需要考虑对其进行重构。对于无依赖项，需要判断它是必需的代码段还是无用的代码段。

为本文的 C＃项目创建代码图，分别检测在项目的不同层次（架构、设计）上是否存在不良的依赖关系。图 1 给出了 VS 创建的代码图以及运行的依赖关系分析器。由于本项目中不存在循环依赖，因此代码图中没有呈现。除了代码图，代码度量值中类耦合度、继承深度亦可帮助识别并度量代码中存在过度依赖关系的设计 TD，本文将在 4.2 节的代码度量值部分具体介绍。

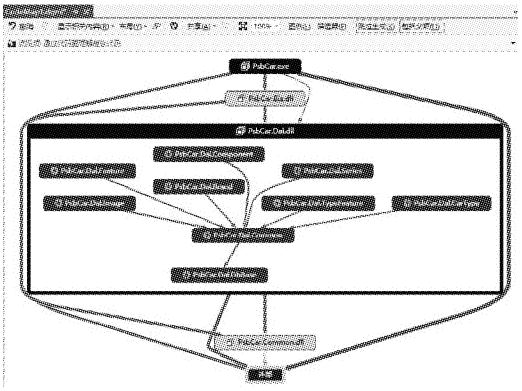

图 1　代码图及分析器

（2）代码与预期架构冲突

VS 的层关系图是对软件项目中的高级结构（模块、组件或功能单元）及其相互作用的展示。通过层关系图可以自动检验项目的代码是否与设计的系统架构冲突。如果存在冲突，则提示验证错误以及错误出现的原因，如无效的依赖关系、依赖于禁止的命名空间等。导致冲突的原因有很多，例如，修改设计的架构时未对相应的代码做出一致的更改，将系统组件分配到错误的层，与预期设计相冲突的依赖关系等。

使用层关系图验证系统架构，有利于架构设计师和开发者在开发过程中尽早发现潜在的架构技术债务，进而通过重构或重写等方法更新代码以使其符合预期的架构。

（3）编码规则冲突

编码规则是为了增强代码的稳定性、可读性、可维护性并提高编码质量与效率而设置的标准。VS 中可自行选择一系列内置的编码规则集，通过运行代码分析，能检测并以列表形式标识出在应用程序中与编码规则冲突的问题，并给予一些修复建议。例如，在 VS 的编码规则集中，“基本的设计准则规则”能识别项目中存在的一些影响可理解性和可维护性的缺陷，如标示符的不规范拼写、意义不明确的命名等。图 2 是对 C＃运行代码分析后识别出的编码规则冲突。

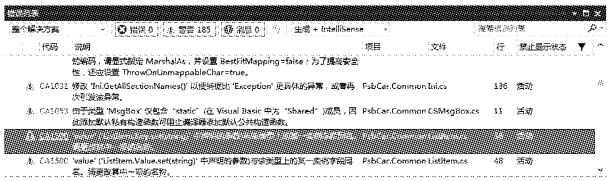

图 2　编码规则冲突

（4）重复代码

重复代码是项目中经常存在的一种代码技术债务，降低了代码的可变性和重用性。在 VS 中，通过“分析代码克隆”可检测部分范围或整个项目中的重复代码，结果依据相似性程度展示。图 3 给出了“分析克隆代码”功能检测到的 C＃项目中的重复代码。VS 能查找到的代码重复不仅限于完全相同的代码段，还包括改写程度不超过 40％的相似代码段，如插入、删除或重新排列部分语句，更改变量以及参数的命名。

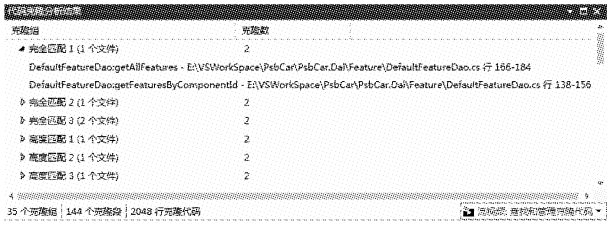

图 3　重复代码检测

在对代码进行更新或修复前，检测其是否存在重复片段，以避免由于不一致的更改而造成错误；同时，开发者可以考虑是否对重复代码进行重构或清理。

4.2　VS 实现的技术债务管理活动（RQ2）

VS 实现了对技术债务的识别、度量、优先级排序、预防、监控、偿还和交流，已经很大程度地突破了大部分工具的局限。对某些类型的技术债务不仅可以定性地管理，还可以定量地检测和分析，如识别并度量不良的依赖关系。表 2 列出 VS 中可用于技术债务管理的具体方法。

表 2　技术债务管理活动和方法

技术债务管理活动	方法	Visual Studio 实例功能
识别	代码分析	检测违反编码规则的代码、代码克隆；依据代码度量值识别技术债务
	依赖性分析	代码图及分析器识别不良依赖关系
	图验证	层关系图验证代码与预期架构间的冲突
度量	计算代码相关度量值	代码度量值、代码覆盖率
优先级排序	按维护难易程度排序	排序代码度量值，优先处理最难维护的代码段
预防	改善开发过程	“智能输入提示”可跟踪开发，提供修复和重构建议
监测	质量属性的变化	查看“代码覆盖率”的历史结果，了解变化趋势
	追踪技术债务的传播	根据代码图显示的依赖关系了解技术债务的传播
偿还	重构	封装字段、移除参数、提取方法、重命名等重构
	自动化	“IntelliTest”功能可为代码自动生成单元测试
交流	依赖可视化	代码图可用不同颜色标注不良的依赖关系
	代码度量值可视化	不同颜色标识代码可维护性的难易程度

针对表中一些不易理解、难以简要说明的方法，下面将进行详细的描述，包括 5 种代码度量值和 5 种可用于偿还技术债务的重构方法。

(1)代码度量

代码度量能够以量化的方式使开发者更好地了解当前代码的质量和可维护性，帮助识别项目中潜在的技术债务。VS 提供了对 5 种代码度量值的计算，分别是圈复杂度、继承深度、类耦合度、可维护指数和代码行数，如图 4 所示。开发者还可以设置筛选器来选出所关注的代码度量值结果，或对所关注的数据列进行排序，以此来决定亟需优先解决的潜在问题，如筛选出可维护指数低于 20 的代码片段。

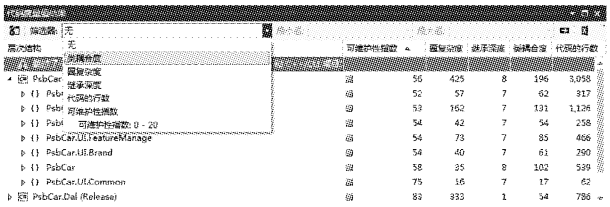

图 4 技术债务相关代码的度量值

1)圈复杂度：通过计算代码段中存在的不同分支路径数，衡量代码在结构上的复杂程度，可用于识别度量复杂代码类型的技术债务。具有较大圈复杂度值的代码段不仅难以维护，还需要更多的测试才能实现良好的代码覆盖率。

2)可维护指数：基于操作数、运算符、圈复杂度等多个指标计算出的一个 0～100 间的指数值，用于衡量代码可维护的容易度。值越大表示代码越容易维护，可以帮助开发者发现项目中的低质量代码段。

3)继承深度：测量对象到继承层次结构的根的深度数目。值越大表示继承层次结构越深，继而越难理解和维护特定对象的设计情况，过度的继承更易引入对象之间复杂且混乱的设计依赖关系。

4)类耦合度：衡量唯一类与其他类型之间的耦合程度，通过引用类的数量来标识。其值越小，意味着更改这部分代码对其他类型的影响越小。良好的软件设计要求类型和方法应具有较高的内聚度和较低的耦合度，有利于代码的重用和维护。

5)代码行数：计算的可执行代码的行数，不包括括号、注释、声明等。如果一个类或方法的代码行数过多，表示类或方法可能过于复杂，所执行的功能过多，增加了出现技术债务的可能性，加大了重用和维护难度，可采取重构等方式对功能点进行拆分。

(2)重构

重构是在不更改软件外部行为的前提下，对代码内部结构进行修改，进而改进软件质量，使其具有更好的可扩展性、可重用性和可维护性。VS 能够实现对代码的自动重构，但由于重构的逻辑与编码语言的语法有关，因此其对不同语言所提供的重构方法有所差异。对于本文采用的 C＃项目，VS 提供的可用于对已识别技术债务进行偿还的重构方法主要有以下几种。

1)提取方法：从现有代码段中选取代码构成新方法，通过调用的方式更好地进行组织以获得精炼的代码。对于项目中复杂的类或方法，由于其承担的执行工作量过大，难以维护和理解，通过提取方法对功能进行分离细化，可提高代码的可重用性。

2)重命名：对一些意义不明确、不能很好表达标识符用途的名称进行重命名重构，使其符合命名规范并且更易于理解。

3)封装字段：对现有字段的快速封装，建立 get 和 set 访问器，通过这些函数实现对字段的合理访问。

4)提取接口：当有多个类、结构或接口都使用同一子集时，创建及引入新的接口来解除这种依赖关系。提取接口重构有利于软件项目的模块化，降低代码的耦合程度，提高代码的可扩展性和可维护性。

5)删除参数：将多余的参数从方法中移除，并在所有的调用位置删除参数，形成新的声明。在实际开发过程中，程序员往往不愿意去掉多余的参数，以备后续之用，但这种冗余的设计会为开发者增添额外的负担去思考这些参数的意义以及调用时需要传递的内容。

4.3 对比 VS 和其他工具的技术债务管理能力(RQ3)

表 3 对比展示了 VS 以及 4 种专门的技术债务管理工具在可管理的技术债务类型、具体实例以及支持的管理方法等方面的能力。

表 3 VS 与 4 种工具的技术债务管理能力对比

技术债务类型	技术债务实例	识别	度量	优先级排序	表示/文档化	预防	监测	偿还	交流
架构 TD	不良的架构依赖	1							1
	代码与预期架构冲突	1							1
设计 TD	设计冗余	1						1	
	复杂的类或方法	1	1	1				1	
	不良的设计依赖	1,4	1	1	4		4	1	1,4
代码 TD	代码重复	1	1	1					1
	低质量代码	1,2,3	1,2,3	1,3			2		2,3
	编码规则冲突	1,5				1	5	1,5	
	复杂的代码	1	1	1					
测试 TD	低代码覆盖率	1,5	1,5	5			1,5	5	5
	缺少测试	1						1	

注：1 代表 Visual Studio，2 代表 Software maps tool，3 代表 Technical Debt Evaluation plugin for SonarQube，4 代表 DebtFlag，5 代表 Sonar TD plugin

其中，Software maps tool 和 Technical Debt Evaluation plugin for SonarQube 均用于管理项目中出现的低质量代码(代码 TD)；DebtFlag 能够识别、监测不良的设计依赖(设计 TD)并以文档化形式表示；Sonar TD plugin 能够管理编码规

则冲突（代码 TD）以及低代码覆盖率（测试 TD）。相比这些工具，VS 能够管理的技术债务类型和实例更加广泛，涉及架构、设计、代码、测试，包括不良的架构依赖、复杂的类/方法、重复代码、低代码覆盖率等 11 种具体技术债务实例。对可识别的 11 种技术债务实例中的大部分可进一步实现度量、优先级排序、偿还以及交流等管理活动。但是，针对一些具体技术债务实例，VS 的管理能力有一定不足，比如，相比于 Sonar TD plugin，缺少对低代码覆盖率的优先级排序、偿还和交流。

5　讨论

理解技术债务概况对开发团队中的不同利益相关者有重要的指导意义。迫于市场压力，开发团队需要不断为已有项目开发新版本。团队的架构设计师在理解现有项目的架构时发现在长期的维护过程中已经引入了大量的技术债务，并意识到只有预先对技术债务进行偿还才能支持新功能。因此，可以通过执行一系列的代码分析，产生相关度量值和依赖关系，再依据这些结果制定对代码的重构计划。项目经理在了解技术债务对系统演变的阻碍后，在制定项目解决方案时也会充分考虑如何着手偿还这些技术债务，同时平衡项目交付时间和预算要求。对于开发者而言，其在扩展现有项目功能时，可以有意识地采取规范的编码策略，进而改善所交付代码的质量。

在实际开发中，引入技术债务对项目的影响程度常常被低估。项目中存在各种技术债务（如设计之间紧密的耦合，接口没有被明确定义，一个组件负责过多的功能，缺少单元测试，代码没有被封装[6]），可以想象每次重构和扩展的困难程度，而且极易发生错误，极端情况下甚至需要重写全部代码或放弃对项目的扩展。如果开发团队意识到存在的技术债务对系统的不良影响，定期采取措施进行监控并及时偿还当前存在的技术债务，未来对软件的扩展将会更加容易。开发者也可以在开发新功能的同时更加关注编码的规范性，尽量避免不良的设计决策，进而提高软件的质量。

Visual Studio 在协助开发团队管理技术债务时，避免了已有工具存在的一些局限性：1）支持 C＋＋，C＃，F＃，Visual Basic 等多种编程语言项目；2）管理的技术债务类型及具体实例更加广泛；3）支持不限于特定的某一项技术债务管理活动；4）分析技术债务的方法更加多样，如代码分析、依赖性分析、计算代码度量值；5）实现对一些类型的技术债务实例的自动偿还，提供多种重构方法。

尽管 VS 在管理技术债务方面已经体现出了较大的优势，但鉴于对期望技术债务管理工具的需求，其仍具有以下不足：1）不能从经济视角衡量管理技术债务的成本[26]。相比于代码度量值等数据，产品经理更加关注偿还技术债务所需的时间或成本。VS 没有提供对在不同时间点解决同一技术债务将产生的不同经济后果（如时间、代码量）的分析比较。2）管理技术债务的功能分散，易用性不够强。VS 中各种与提高代码质量相关的功能分布在不同位置，缺少专门的菜单项，需要开发者对技术债务有一定了解。3）对技术债务实例不能实现完整的管理流程。有效的管理方式是能够对每种技术债务实现从识别定位、度量分析、按序偿还到监控预防的系统管理过程。充分了解 VS 的优势和不足可以为开发更合适的技术债务管理工具提供支持。

本文的探究工作虽然对提高软件开发的质量有一定指导意义，但仍然存在一些不足，比如只考虑了 VS 对以 C＃为编程语言的项目的管理能力，对其他程序语言的技术债务管理的支持程度以及结论的适用性有待考查。

结束语　本文通过分析技术债务的产生原因、类型以及它对软件质量的影响，阐明了对技术债务进行管理的必要性，并进一步介绍了技术债务管理活动和方法，以帮助软件开发团队理解技术债务及其管理的概况。同时，针对目前技术债务管理工具支持的技术债务类型、编程语言单一等局限性，结合主流集成开发环境 Visual Studio 的实际广泛应用，通过一个C＃项目实际探究它对软件中可能存在的架构、设计、代码和测试这 4 种主要类型的技术债务能够提供的管理技术支持，同时对比它与 Software maps tool，Technical Debt Evaluation plugin for SonarQube，DebtFlag，Sonar TD plugin 这 4 种专门的技术债务管理工具在技术债务管理能力上的优势与不足。实验结果显示，Visual Studio 能够采取多种管理方法对架构、设计、代码以及测试中出现的 11 种技术债务实例实现不同程度的识别、度量、优先级排序、监测、偿还、交流等管理活动。Visual Studio 作为主流集成开发环境，相较于已有技术债务管理工具，尽管对个别具体技术债务实例的管理程度略逊于专门的工具，但对项目中存在的广泛的技术债务类型具有更好且更全面的管理能力。

下一步将继续探究 Visual Studio 插件对技术债务管理的支持程度；同时进一步对比其他主流集成开发环境的技术债务管理能力，如 Eclipse，IntelliJ IDEA；最后，研究并开发基于 Visual Studio 的技术债务管理插件，集成 Visual Studio 已有功能并完善技术债务管理所需的其他功能。

参 考 文 献

[1] BROWN N，CAI Y，GUO Y，et al. Managing technical debt in software-reliant systems[C]//Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. New York，USA：ACM，2010：47-52.

[2] CUNNINGHAM W. The WyCash portfolio management system [J]. ACM Sigplan Oops Messenger，1992，4(2)：29-30.

[3] AMPATZOGLOU A，AMPATZOGLOU A，CHATZIGEORGIOU A，et al. The financial aspect of managing technical debt：A systematic literature review[J]. Information and Software Technology，2015，64(C)：52-73.

[4] ALLMAN E. Managing Technical Debt[J]. Queue，2012，10(3)：50-55.

[5] LI Z，AVGERIOU P，LIANG P. A systematic mapping study on technical debt and its management[J]. Journal of Systems and Software，2015，101：193-220.

[6] SURYANARAYANA G，SAMARTHYAM G，SHARMA T. Refactoring for software design smells：Managing technical debt [M]. San Francisco，USA：Morgan Kaufmann，2014.

[7] LIM E，TAKSANDE N，SEAMAN C. A Balancing Act：What Software Practitioners Have to Say about Technical Debt[J]. IEEE Software，2012，29(6)：22-27.

[8] ALVES N S R，MENDES T S，DE MENDONÇA M G，et al. Identification and management of technical debt：A systematic mapping study[J]. Information and Software Technology，2016，70：100-121.

（下转第 40 页）

tions to the Web[M]// Information Technology and Open Source Applications for Education, Innovation, and Sustainability. Springer, 2014: 229-243.

[11] WANG S T, et al. Development of Web-Based Remote Desktop to Provide Adaptive User Interfaces in Cloud Platform[J]. Int'l J. Computer, Information, Systems and Control Eng., 2014, 8(8): 1195-1199.

[12] CHEN B, HSU H, HUANG Y. Bringing Desktop Applications to the Web[J]. IT Professional, 2016, 18(1): 34-40.

[13] VAHDAT A, ANDERSON T, DAHLIN M, et al. Webos: Operating System Services For Wide Area Applications[C]// Proceedings of the Seventh IEEE Symposium on High Performance Distributed Systems. 1997: 52-63.

[14] DAN S, JAMES C, et al. A way forward: enabling operating system innovation in the cloud[C]// Proceedings of the 6th USENIX conference on Hot Topics in Cloud Computing (HotCloud'14). 2014.

[15] RAFAEL M, RUBÉN S, et al. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures[J]. IEEE Computer, 2012, 45(12): 65-72.

[16] FABIO P, PETER B, et al. Toward a Cloud Operating System [C]// Proceedings of the 2010 IEEE/IFIP Network Operations and Management Symposium Workshops (NOMS Workshop). 2010.

[17] JAFFER S, KEDIA P, BANSAL S. Improving Remote Desktopping through Adaptive Record/Replay[C]// Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environment (VEE 2015). 2015: 161-172.

(上接第 21 页)

[9] BOHNET J, DÖLLNER J. Monitoring code quality and development activity by software maps[C]// Proceedings of the 2nd Workshop on Managing Technical Debt. New York, USA: ACM, 2011: 9-16.

[10] BARTON B, STERLING C. Manage project portfolios more effectively by including software debt in the decision process[J]. Cutter It Journal, 2010, 23(10): 19-24.

[11] HOLVITIE J, LEEPPÄNEN V. DebtFlag: Technical debt management with a development environment integrated tool[C]// Proceedings of the 4th International Workshop on Managing Technical Debt. Piscataway, USA: IEEE, 2013: 20-27.

[12] GARCIA J, POPESCU D, EDWARDS G, et al. Identifying architectural bad smells[C]// Proceedings of the 13th European Conference on Software Maintenance and Reengineering. Piscataway, USA: IEEE, 2009: 255-258.

[13] SLINGER S. Code Smell Detection in Eclipse[J]. Delft University of Technology, 2005, 10(1): 78-83.

[14] MORGENTHALER J D, GRIDNEV M, SAUCIUC R, et al. Searching for build debt: Experiences managing technical debt at Google[C]// Proceedings of the Third International Workshop on Managing Technical Debt. Piscataway, USA: IEEE, 2012: 1-6.

[15] GREENING D R. Release Duration and Enterprise Agility[C]// Proceedings of the 46th Hawaii International Conference on System Sciences. Piscataway, USA: IEEE, 2013: 4835-4841.

[16] ZAZWORKA N, IZURIETA C, WONG S, et al. Comparing four approaches for technical debt identification[J]. Software Quality Journal, 2014, 22(3): 403-426.

[17] GUO Y, SEAMAN C. A portfolio approach to technical debt management[C]// Proceedings of the 2nd Workshop on Managing Technical Debt. New York, USA: ACM, 2011: 31-34.

[18] ZAZWORKA N, SEAMAN C, SHULL F. Prioritizing design debt investment opportunities[C]// Proceedings of the 2nd Workshop on Managing Technical Debt. New York, USA: ACM, 2011: 39-42.

[19] SCHMID K. A formal approach to technical debt decision making[C]// Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures. New York, USA: ACM, 2013: 153-162.

[20] EISENBERG R J. A threshold based approach to technical debt [J]. ACM SIGSOFT Software Engineering Notes, 2012, 37(2): 1-6.

[21] FERNÁNDEZ-SÁNCYEZ C, GARBAJOSA J, IDAL C, et al. An analysis of techniques and methods for technical debt management: a reflection from the architecture perspective[C]// Proceedings of the Second International Workshop on Software Architecture and Metrics. Piscataway, USA: IEEE, 2015: 22-28.

[22] BOHNET J, DÖLLNER J. Monitoring code quality and development activity by software maps[C]// Proceedings of the 2nd Workshop on Managing Technical Debt. New York, USA: ACM, 2011: 9-16.

[23] LETOUZEY J L. The SQALE method for evaluating technical debt[C]// Proceedings of the Third International Workshop on Managing Technical Debt. Piscataway, USA: IEEE, 2012: 31-36.

[24] HOLVITIE J, LEPPÁNEN V. DebtFlag: Technical debt management with a development environment integrated tool[C]// Proceedings of the 4th International Workshop on Managing Technical Debt. Piscataway, USA: IEEE, 2013: 20-27.

[25] BARTON B, STERKUBG C. Manage project portfolios more effectively by including software debt in the decision process[J]. Cutter IT Journal, 2010, 23(10): 19.

[26] FALESSI D, SHAW M A, SHULL F, et al. Practical considerations, challenges, and requirements of tool-support for managing technical debt[C]// Proceedings of the 4th International Workshop on Managing Technical Debt. Piscataway, USA: IEEE, 2013: 16-19.