

**软件体系结构第二次作业**

Design Rule Spaces A New Model for representing and analyzing software architecture

精读报告

|  |  |  |
| --- | --- | --- |
| 小组成员： | ZY2006109 | 姬轶 |
| 小组成员： | SY2006316 | 曲卓涵 |

北京航空航天大学

**2021 年03月**

精读报告

|  |  |  |
| --- | --- | --- |
| 题目 | Design Rule Spaces A New Model for representing and analyzing software architecture | 1 |
| 中文题目 | 设计规则空间集(DRSpace)：一种表示和分析软件体系结构的新模型 | 5 |
| 出处 | Cai Y, Xiao L, Kazman R, et al. Design rule spaces: A new model for representing and analyzing software architecture[J]. IEEE Transactions on Software Engineering, 2018, 45(7): 657-682. | 4 |
| 要解决的问题 | 现有软件架构模型对于模块的划分不足以建立起代码缺陷与软件架构之间的联系，亟需一种新型的、可重叠的划分模型，以便更好地从架构角度来维护软件质量。具体问题分为以下四个部分：  (1)、如果一个在架构上有影响的文件(设计规则)容易出现bug，那么其设计空间中的文件是否也可能出现bug，并且比其他随机设计空间更容易出现bug?这个问题的答案将帮助我们理解设计规则对架构的影响。  (2)、容易产生错误的文件在架构上连接到什么程度?如果大部分容易出错的文件被捕获在几个架构根中，这就意味着体系结构对软件质量的重大影响。  (3)、在项目发展过程中，这些主要根源是长期和持久的吗?这个问题的答案将帮助我们理解架构的影响是否持久。  (4)、这些根源是否包含可能导致bug倾向的架构缺陷?最后，我们想了解这些容易出错的文件背后的架构问题是什么。 | 10 |
| 技术路线 | 1. 建立完整的DRSpaces，捕获所有源文件及其依赖项，用以表征软件系统的体系结构。 2. 伪代码1步骤① 3. 伪代码1步骤② 4. …… 5. 建立BugSpace，即软件系统中容易出错的文件集 6. 细节1 7. 细节2 8. …… 9. 检测ArchRoots 10. 伪代码2步骤① 11. 伪代码2步骤②   …… | 10 |
| 关键问题、解决方案 | 首先需要对本文几个核心名词进行解释：  **设计规则(Design Rule)：重要的设计决策，这些决策将系统的其他部分解耦为相互独立的模块。作为解耦接口，设计规则支持模块内部的相互依赖和模块之间的独立。只要遵守和维护既定的设计规则，独立模块的设计、实现和改进不应影响其他模块。**  **设计规则空间集(DRSpace)：由系统文件的子集以及这些文件之间的体系结构连接组成。DRSpace具有以下关键特征:**   1. **DRSpace总是包含一个或多个前导文件，即空间中所有其他文件直接或间接依赖的文件。** 2. **DRSpace捕获文件之间的各种关系。这些关系可以是文件之间的结构依赖关系或演化耦合关系。** 3. **使用DRH算法对每个DRSpace进行集群，根据选择的一种或多种关系类型，自动将引导文件和模块显示在DRSpace中。**   **架构根(ArchRoots)：连接系统中最容易出现错误的文件的最小DRSpace集来实现的。**  本文使用设计结构矩阵(DSM)为软件架构建模。在DSM中，列和行表示系统的设计参数(顺序相同)。DSM的单元代表了它们之间的关系。非对角单元格表示行上的文件与列上的文件之间的依赖关系。对角单元格表示自依赖性，我们通过文件的行/列索引来注释这些单元格。  设计规则层次(DRH)算法，将源文件进行聚类，使其架构角色一目了然。使用DRH算法聚类的DSM有三个关键特征。首先，将设计规则和模块按照层次结构进行排列，设计规则在上层，模块被设计规则解耦在下层。其次，低层的模块依赖于高层的模块，但反之亦然。第三，同一层次的模块是相互独立的。  架构决策一旦在源代码中实现，就会变得很难理解、很难修改，并且可能会因为不断的发展而逐渐衰败。因此架构选择是源代码中技术债务的头号来源。  同时技术产生了限制性的集群:也就是说，一个软件单元只能属于一个单独的集群。然而，在现实中，一个文件扮演多个角色并参与其中是很自然的。  具体的问题与解决方案如下所示：   1. DRSpaces的生成。   问题描述：DRSpace可以以前导文件为基础而生成，但是前导文件可以是一个或者多个，因此在生成DRSpace的过程中面临算法复杂度过高的问题。  解决方案：选择软件系统中每个单独的源文件作为前导文件，生成DRSpace。这样既简化了DRSpace的生成，同时还便于探究每个源文件作为主导文件时的影响。   1. 影响因素的排除。   问题描述：在探究DRSpace的bug倾向性程度时，会受到DRSpace中的文件数、不同文件的代码行数等因素的影响。  解决方案：同时使用dsb（DRSpace中出现bug的文件比例）和bsc（DRSpace中bug文件占全部bug文件的比例）来分析DRSpace的bug倾向性程度；进行LOC规范化，消除代码行数的影响。 | 10 |
| 验证手段 | 实验对象：选取15个Apache开源项目作为评估对象。这些项目的规模、应用领域、开发时间等特征都有所不同。  实验目标：探究如下四个问题：   1. 前导文件的错误是否导致其DRSpace中的文件更容易出现错误？ 2. 容易产生bug的文件在架构上是否具有紧密联系？ 3. 在项目推进过程中，ArchRoots会长期保持不变吗？ 4. ArchRoots是否表示其中的文件具有bug倾向？   实验指标：   1. dsb：DRSpace中出现bug的文件比例 2. bsc：DRSpace中bug文件占全部bug文件的比例 3. Length：项目的开始、结束时间以及持续时长 4. Releases：版本数量 5. Commits：提交数量 6. Issues：问题数量 7. Files：文件数量   实验结果：   1. 如果一个设计规则（前导文件）容易产生错误，那么在其DRSpace中聚合的文件也容易产生错误。 2. 容易出现bug的文件往往在架构上彼此关联 3. 一般来讲，每个项目都会包含1-5个稳定的ArchRoot。   每个ArchRoot中，都观测到了一些反复出现的bug，即ArchRoot往往预示着这些文件的高bug率。 | 10 |
| 论文得出的结论 | 本文提出了一种新的软件体系结构模型DRSpace，它可以捕获复杂软件系统中的多个重叠的设计空间。  本文通过在DRSpace中检测ArchRoot，并进行分析与验证，揭示了体系结构对软件质量的影响。即：   1. 容易产生bug的规则会使其DRSpace中的文件更容易出错。 2. 容易出现bug的文件在架构上具有显著的关联，并且会持续一段时间。 3. 大多数软件系统的bug文件集中在5个ArchRoot中。 | 10 |
| 主要贡献 | 1. 提出了DRSpaces的概念，定义了一种可重叠的新型软件架构模型。 2. 提出了ArchRoots的概念，建立了代码缺陷与软件架构规则之间的一种联系，可以引导开发者重点关注具有bug倾向的ArchRoots，力求从架构根源上减少缺陷以及缺陷的传播。 3. 提出了一种寻找ArchRoots的算法，便于开发者追溯bug的架构来源。 | 10 |
| 最可借鉴之处 | 先前的研究往往将软件系统中的模块划分为不重叠的集合，而本文从设计规则这一角度入手，创新性地提出了一种可重叠的软件架构表征方法，由此更为显著地揭示了软件体系结构与软件质量之间的关联。 | 10 |
| 存在的问题 | 1. 语言局限性：本文研究的15个开源项目都是用Java实现的，并且前期调研中研究的的工业项目都是采用Java或C++实现的。因而本文提出的DRSpace对于采用面向对象思想实现的软件系统具有较好的效果，然而对于面向过程或其他思想的软件系统不一定具备适用性。 2. 预测局限性：DRSpace建模和分析采用逆向工程的回溯方法，没有对该方法的预测能力进行任何调查，例如，文件之间有缺陷的体系结构连接可能会如何“增长”，或者哪些DRSpaces在未来可能仍然容易出现bug。 3. 有效性：ArchRoot检测算法的准确性过于依赖可用数据质量，如果跟踪和版本控制系统存在错误，这对识别易错性的架构根源的有效性构成了外部威胁。 4. DRSpace中对于两个文件的隐式关联（演化耦合）的捕获，是通过分析软件提交记录中同时修改的文件来进行判断的。考虑到开发过程的不确定性，这种方式可能无法捕获所有隐式关联，甚至捕获到错误的关联。 5. 在DRSpace的生成上，本文是以每个文件作为前导文件来生成DRSpace的，这对于某些设计模式并不能生成合理的DRSpace。 | 15 |
| 重要参考文献 | 1. C. Y. Baldwin and K. B. Clark, Design Rules, Vol. 1: The Power of Modularity. Cambridge, MA, USA: MIT Press, 2000. 2. S. Huynh, Y. Cai, and K. Sethi, “Design rule hierarchy and analytical decision model transformation,” Drexel University, Philadelphia, PA, USA, Tech. Rep. DU-CS-08-04, Nov. 2008, 3. S. Huynh, Y. Cai, and K. Sethi, “Design rule hierarchy and model transformations,” Proc. Presented Student Res. Forum 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng., Nov. 2008. 4. A. Corazza, S. D. Martino, V. Maggio, and G. Scanniello, “Investigating the use of lexical information for software system clustering,” in Proc. 15th Eur. Conf. Softw. Maintenance Reengineering, Mar. 2011, pp. 35–44. 5. R. Naseem, O. Maqbool, and S. Muhammad, “Improved similarity measures for software clustering,” in Proc. 15th Eur. Conf. Softw. Maintenance Reengineering, Mar. 2011, pp. 45–54. 6. T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Where the bugs are,” in Proc. 13th ACM SIGSOFT Int. Symp. Softw. Testing Anal., Jul. 2004, pp. 86–96. 7. R. Schwanke, L. Xiao, and Y. Cai, “Measuring architecture quality by structure plus history analysis,” in Proc. 35rd Int. Conf. Softw. Eng., May 2013, pp. 891–900. 8. Y. Cai, H. Wong, S. Wong, and L. Wang, “Leveraging design rules to improve software architecture recovery,” in Proc. 9th Int. ACM Sigsoft Conf. Quality Softw. Archit., Jun. 2013, pp. 133–142. 9. S. Wong, Y. Cai, M. Kim, and M. Dalton, “Detecting software modularity violations,” in Proc. 33rd Int. Conf. Softw. Eng., May 2011, pp. 411–420. 10. S. Wong, Y. Cai, G. Valetto, G. Simeonov, and K. Sethi, “Design rule hierarchies and parallelism in software development tasks,” in Proc. 24th IEEE/ACM Int. Conf. Autom. Softw. Eng., Nov. 2009, pp. 197–208. 11. R. Mo, Y. Cai, R. Kazman, and L. Xiao, “Hotspot patterns: The formal definition and automatic detection of architecture smells,” in Proc. 15th Work. IEEE/IFIP Int. Conf. Softw. Archit., May 2015, pp. 51–60. 12. N. Brown, et al., “Managing technical debt in software-reliant systems,” pp. 47–52, 2010. [Online]. Available: http://doi.acm. org/10.1145/1882362.1882373 | 5 |

小组成员分工情况

1. 初步选出几篇满足条件的文章进行abstract、introduction阅读。
   1. 姬轶：筛选论文、阅读摘要并进行比对。
   2. 曲卓涵：筛选论文、阅读摘要并进行比对。
2. 选择较为贴合课程且有一定影响力的论文进行精读。
   1. 姬轶：精读论文、完成精读论文报告并整理合并归档。
   2. 曲卓涵：精读论文、完成精读论文报告。
3. PPT制作。
   1. 姬轶：初版完成后PPT修改。
   2. 曲卓涵：负责整体制作。