第 34 卷 第 1 期 计 算 机 工 程 2008 年 1 月
Vol.34 No.1 Computer Engineering January 2008
·软件技术与数据库· 文章编号：1000—3428(2008)01—0094—03 文献标识码：A 中图分类号：TP311.5

软件体系结构层面的影响分析

孙 霞[1,2]，杨 芸[3]
(1. 浙江工业大学计算机学院，杭州 310014；2. 浙江嘉兴职业技术学院，嘉兴 314000；3. 浙江行政学院信息管理部，杭州 310012)

摘 要：基于场景的软件体系结构层面的系统可维护性评估主要依赖于情景的萃取和体系结构层面的分析技术。该文研究了软件体系结构层面的影响分析技术，分析该技术目前面临的挑战，并提出了相应的解决方案，试图通过对该技术的研究，进一步提高软件体系结构层面质量评估方法的实用性，增强系统的可维护性。

关键词：影响分析；软件体系结构；可维护性

Impact Analysis in Software Architecture Level

SUN Xia[1,2], YANG Yun[3]
(1. College of Computer, Zhejiang University of Technology, Hangzhou 310014; 2. Jiaxing Vocation Technology College, Jiaxing 314000;
3. Information & Management Department, Zhejiang Administrative Institute, Hangzhou 310012)

【Abstract】Software-architecture-level maintenance evaluation based on scenario mainly relies on two technologies: elicitation of the scenarios and impact analysis in architecture level. This paper focuses on the impact analysis in architecture level. It studies the challenges that impact analysis faces, and presents the solution. The study in this technology can improve the efficiency of the many software-architecture-level evaluation methods, and can improve system maintenance.

【Key words】 impact analysis; software architecture; maintenance

1 概述

变更在整个软件生命周期中发生频繁且不可避免，因此，系统的维护在软件开发中越来越重要。系统的可维护性从根本上受制于软件体系结构，10 多年来，研究者们提出了多种软件体系结构层面的系统可维护性评估方法，试图在系统开发的早期阶段通过设计灵活的体系结构来保证软件的可维护性[1-2]。多数体系结构层面的评估方法都建立在情景分析技术之上，虽然在情景萃取方法、可维护性评估技术的细节上有所不同，但大都遵循相同的思路，即利用情景来描述系统可能发生的变更(SAAM[2]中的间接情景、ALSM[3]中的变更情景等)，确定这些可能的变更给系统带来的影响，然后综合所有情景的分析结果，得到软件体系结构总体的可维护性或灵活性。基于情景的软件体系结构可维护性评估的过程为：(1)情景提取；(2)确定每个变更情景对系统的影响；(3)根据情景的重要性或不同属性，综合每个情景的影响分析结果，得出系统可维护性的总体结论。从上述过程可以看出，基于情景的可维护性评估方法主要依赖 2 方面技术：对软件系统可能发生的变更活动的成功预测和体系结构层面上的变更影响分析技术。目前大多数体系结构层面的评估方法都将注意力集中到解决情景提取上，对第 2 个问题的关注相对较少。影响分析是一项相对成熟的技术，在软件演化、重构、调试等领域中已有较多研究，但传统的影响分析基本集中在源代码级别上，与软件体系结构层面的分析有较大的差别，不能单纯地将传统影响分析技术移植到体系结构级别上。

本文在分析传统影响分析技术的基础上，指出软件体系结构层面的影响分析具有自己的特点，对其进行深入研究能够极大地提高软件体系结构层面可维护性分析方法的实用性。

2 基本概念

2.1 软件体系结构

软件体系结构 HAI 没有统一的定义，但研究界不缺乏有关定义，例如 SEI 的网站上收集了上百个关于体系结构的定义。虽然这些定义各有侧重点，但几乎都认为软件体系结构描述了软件系统在较高层次上的整体结构，组件之间的交互是构成软件体系结构的基本元素。基于这种认识，本文采用文献[4]中的定义：软件体系结构指的是系统的结构，这些结构由软件元素、这些元素的外部可见属性以及元素之间的相互关系构成。

软件体系结构的描述方法很多，可以简单地分为 3 类：

(1)开发者临时定义的各种简单符号，用来解释软件系统的总体结构，即常说的 box-and-line 表示；

(2)用形式化的符号对软件体系结构进行描述，如众多的 ADL 语言；

(3)半形式化的图形语言，如 UML。

2.2 影响分析[3]

影响分析是为了确定某个变更对整个系统所产生的影响，以下几个概念在影响分析中经常涉及到。软件生命周期对象(Software Life-cycle Objects, SLOs)，也叫作软件产品或者软件制品，是软件开发中产生的成果，例如一条需求，一个类或一个组件。生命周期对象通过关系网相互联系，关系网可将同类型的生命周期对象联系在一起，也可将不同的生命周期对象联系在一起。例如，可将软件体系结构中 2 个组

作者简介：孙 霞(1971－)，女，讲师，主研方向：数据压缩；杨 芸，硕士

收稿日期：2007-02-22 **E-mail：**xueqin_jx@163.com

件联系在一起，也可将软件体系结构中的组件与软件实现中的类联系在一起，这 2 种关系不相同，在影响分析中起着不同的作用。

影响分析主要包括依赖性分析和追踪性分析。依赖性分析在同一个抽象级别上进行。软件体系结构层面上的分析主要分析组件以及组件之间的交互关系，属于依赖性分析。追踪性分析则关注整个开发周期内多种软件周期对象之间的关系，例如体系结构与系统实现之间的关系。

在影响分析中主要有如下集合值得注意：(1)系统集，即系统中所有软件周期对象的集合；(2)初始影响集，即变更发生时预计直接受影响的集合。估计受影响集是指所有由于变更的实施而估计可能受到影响的集合。初始影响集是估计受影响集的真子集，对初始影响集使用变更传播规则得到受影响集。实际受影响集是指实现变更后，系统中实际上受到影响的对象的集合。

变更影响分为主要影响和次要影响。主要影响也叫直接影响，是分析变更活动如何影响系统时确定的软件生命周期对象；次要影响也叫间接影响，包括副作用和波及效应。

3　软件体系结构层面的影响分析

传统的影响分析技术以函数和变量间的控制流和数据流为基础，最开始用于编译器的优化以确定某个代码重构是否安全。随着软件复杂度的提高和技术的发展，随着以体系结构为中心，渐增式的开发、基于组件的开发等新开发方法的出现，影响分析技术也突破了原来的适应范围。体系结构层面的影响分析和传统的影响分析存在如下不同：

(1)分析的对象不同，从源代码转移到了抽象的体系结构模型；

(2)分析的目的不同，传统的影响分析通常需要精确定位变更影响的目标，以便制定变更实施计划；体系结构层面的影响分析则更侧重变更的影响范围和变更实现的复杂性，以便对体系结构的灵活性作出评估。

影响分析一般分 2 个阶段进行：确定直接影响实体和确定间接影响实体。直接影响的确定主要依靠开发者的经验，很难自动进行；间接影响的范围较广，凭经验或者感觉难以把握，通常借助于自动化的分析方法。

3.1　直接影响分析

直接影响分析需要确定变更活动如何影响软件体系结构。变更往往源自软件需求的变化，属于问题空间，而体系结构属于解空间，将问题空间映射到解空间是一个非常复杂的过程，大部分依靠软件架构师的创造力和经验来进行，不存在通用的工程方法。这种主观性往往使得分析过程、评估过程可重复性差，分析的效果依赖于评估者。如何为这一主观过程提供更多的指导使其更具系统性，是该阶段分析主要面临的问题。

记录软件设计原因被认为能够帮助工程师确定直接影响，文献[5]中设计了一个受控试验来验证软件设计原因对软件影响分析的作用，结果是肯定的。实际上，设计原因一直被视为软件体系结构信息的一部分而受到重视，但这些原因并没有显式记录下来，往往只存在于设计师的头脑中。设计原因未被记录可能是由于记录设计原因有助于理解和交流系统，但对于构造系统并没有直接的利益，再加上开发时间等方面的原因，设计者并不愿意将原因记录下来。本文认为有 2 条途径可以帮助改善设计原因的记录：

(1)格式化记录，有针对性的记录对维护活动有帮助的有关内容，这样既可以控制记录的大小，同时又对记录人员提供了指导；

(2)加强设计原因和设计结构之间的联系。

3.2　间接影响分析

一般假设系统在某个变更发生前处于协调一致的状态，变更活动会打破这种一致性，因此，需要变更系统其他部分来维持这种一致性。变更往往通过关联关系传播到其直接相邻的组件，而被影响到的相邻组件又进一步传播到与其相邻的组件，该过程不断重复，直到系统重新恢复一致性为止。影响分析首先需要确定变更传播的方式（通过何种关联关系进行传播），然后通过找出体系结构中表示该种关联关系的闭包子集来确定受影响集。

大多数体系结构层面的影响分析只考虑直接影响分析，认为间接影响分析可以忽略，或者认为软件体系结构的信息不足以进行间接影响分析。实际上，考虑间接影响可以在任何层面上进行，只不过信息的具体程度不一样，分析方法的精确度不一样。本文从准确性和精确度这 2 个方面来考虑影响分析方法的有效性，下面给出这 2 个概念的定义。

假设体系结构所有元素的集合(即系统集)为X；对某个变更C，分析方法所给出的结果为估计受影响集合，记作E_C；实现变更C时，实际受到影响的集合，即实际受影响集为A_C，则针对变更C，分析方法的准确性f_C为

$$f_C = (E_C \cap A_C) / A_C$$

针对变更C，分析方法的精确性e_C为

$$e_C = (E_C \cap A_C) / E_C$$

针对系统的软件体系结构 A 以及提取的情景集合 S，定义分析方法的准确性为

$$f_{AS} = \Sigma \ f_S / |S|$$

同样，精确性定义为

$$e_{AC} = \Sigma \ e_S / |S|$$

从上述定义可以看出，如果影响分析方法简单地取系统集作为被影响集，即 $E=X$，则分析方法的准确性为 1。如果变更实际影响到了整个系统，即 $A=X$，则不管什么分析方法，其精确性都为 1。实际情况中，取系统集的做法是毫无意义的，影响分析的作用就在于缩小考察的范围，即尽可能精确地指出变更影响的部分。只有精确性和准确性都比较高的方法才具备很好的实用性。有数据表明，受影响的集合往往被高估，因此，提高分析方法的精确性非常重要。但要提高方法的精确性必须依赖组件的精确语意进行推导。

下文将利用这 2 个评价指标来考察影响体系结构层面分析方法的因素。首先考虑最简单的方法，即软件体系结构的 box-and-line 描述，它通常是设计师随手画出的示意性简图。这种体系结构图中，节点表示系统的组件，节点之间的连线表示组件之间存在一定的关联，如图 1 所示。

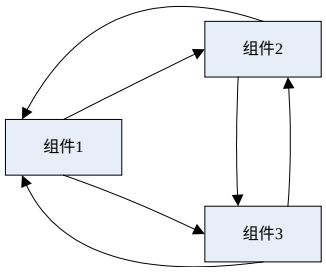

图 1　软件体系结构的 box-and-line 描述

如果简单地利用组件和组件之间的简单关联关系，则对

系统任何部分的变更都会影响到其他部分，如图 1 所示，任何组件的改变都会影响到其他 2 个组件。对系统进行这么粗粒度的影响分析也没有任何实际意义。因此，需要更加详细的信息，以增加方法的精确性、缩小考察组件的范围。

图 2 中增加了有关端口的信息。UML2.0 中为了增加对软件体系结构描述的支持，增加了端口以及连接子的概念。端口分为 input 和 output 两种，一个组件的 input 端口和其他组件的 output 端口通过连接子连接。同时，在组件内部，UML 也提供了相关的描述将组件的 input 端口连接到 output 端口。这些信息描述了组件之间的交互情况，有助于缩小影响分析的范围。

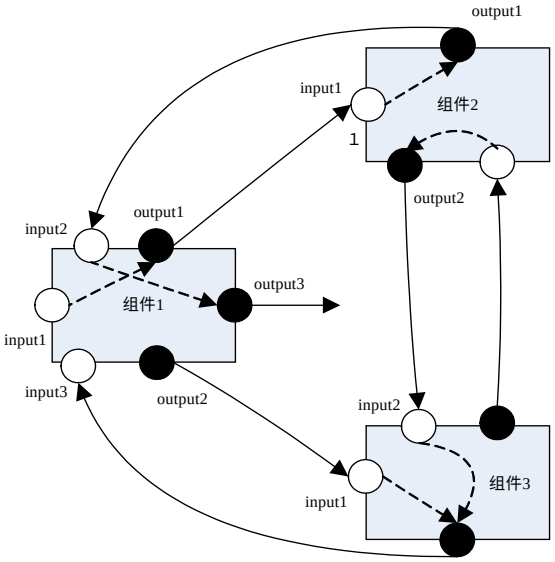

图 2　增加了端口的软件体系结构描述

图 2 是对图 1 增加了端口信息的描述。在图 2 中，变更通过组件 1 的 input 端口 input1 传递进来；而 input1 的变更会影响到 output2 端口，于是变更通过 output2 传播到组件 2 的 input1 端口；组件 2 的 input1 端口通过对 output1 的影响进一步传递到组件 1 的 input2 端口，而 input2 则影响到了 output3，通过 output3 把变更进一步传播。通过对这一部分的分析可以看出，对组件 1 的 input1 进行修改只会影响到组件 2 的部分端口，而不会涉及到组件 3。在图 2 中可以注意到 2 种比较特殊的情况：

(1)input 端口不对 output 端口产生影响，这种情况下对 input 端口的更改只会影响到组件的内部实现，不会进一步传播，例如变更从组件 1 的 input3 传播进来，则 input3 上的变更只会对组件 1 的内部实现产生影响，这种变更不会进一步传播；

(2)组件的 output 端口不受 input 端口的影响，这就意味着只有当组件的某些内部实现发生变动时，该 output 端口才会受到影响，也就是说连接到该端口的组件只受该端口的直接影响，而不会被其他变更影响波及到，例如组件 3 的 output2 端口。

通过端口以及端口的连接关系在保证准确度的前提下能够大大提高变更影响分析的准确度，更重要的是，它不需要额外的信息，标准的 UML2.0 提供了所需的全部信息。在此层面上进行变更影响分析可以花较少的代价获得比较理想的效果。

提高分析的精确性有以下 2 种途径：

(1)从不同角度利用不同的关联关系对体系结构进行分析，取多种分析集合的交集，从而得到比较精确的结果；

(2)进一步提供组件间交互的细节，例如利用对象限制语言(OCL)对交互进行限制，利用这些限制信息进一步限制受影响集合的范围。

第 1 种方法是从横向考虑增加精确度，第 2 种则从纵向加强分析的精确性。第 2 种方法需要较多的设计细节，适合在设计开发的后期，软件系统的行为比较确定的时候进行，在体系结构阶段更倾向于第 1 种方法，例如可以利用体系结构描述的其他视图(部署视图、行为视图等)定义其他类的相关关系，并从中推导影响范围，然后通过这几个视图之间的关系得出最后的受影响集合。

4　结束语

目前大多数体系结构层面的可维护性评估方法实用性都不强，这主要是由这类方法所依赖的 2 种技术的主观性造成的。研究大都围绕如何从大量的情境中得到一个完整而具有代表性的集合展开，如组织尽可能多的软件参与人员进行集体讨论；提供启发式规则等。这些方法在一定程度上增加了方法的系统性。

本文从评估性方法的另一个技术，即基础软件体系结构层面的影响分析出发进行研究，分析了软件体系结构层面影响分析的 2 个阶段，即直接影响确定和波及影响的分析，并分别就这 2 个阶段所遇到的问题和相应的解决方案进入了深入的研究。本文认为，软件体系结构层面的影响分析对进一步提高软件体系结构层面可维护性评估方法的实用性具有重要的作用。

下一步则是将该研究成果用于对软件体系结构评估方法进行改进，并将其用于实际系统，以验证方法的有效性。

参考文献

[1] Kazman R, Abowd G, Bass L, et al. Scenario-based Analysis of Software Architecture[J]. IEEE Software, 1996, 13(6): 47-55.

[2] Bengtsson P O, Bosch J. Architecture Level Prediction of Software Maintenance[C]//Proc. of the 3rd European Conf. on Software Maintenance and Reengineering. [S. l.]: IEEE Computer Society Press, 1999-03.

[3] Jönsson P, Lindvall M. Impact Analysis, Engineering and Managing[M]//Software Requirements. Berlin: Springer-Verlag, 2005.

[4] Bass L, Clements P, Kazman R. Software Architecture in Practice[M]. 2nd ed. Massachusetts: Addison-Wesley, 2003.

[5] Johansson B E, Regnel B. Is a Design Rationale Vital When Predicting Change Impact? — — A Controlled Experiment on Software Architecture Evolution[C]//Proceedings of International Conference on Product Focused Software Process Improvement. Oulu, Finland: [s. n.], 2000.