软件体系结构评估方法的研究*

胡红雷，毋国庆，梁正平，刘秋华
(武汉大学 计算机学院，湖北 武汉 430072)

摘　要: 体系结构评估是软件开发中的一项重要工作,目的是分析体系结构潜在的风险,并检验设计中提出的质量属性需求。介绍并讨论了有代表性的三种评估方法: SAAM, ATAM, ALPSM,在此基础上提出一个概念上的比较框架来分析这些评估方法的相似与差异,并进一步对方法的结合、重用,以及在设计中引入评估及实践中的使用等问题作了探讨。

关键词: 软件体系结构评估; 质量属性; 场景; SAAM; ATAM; ALPSM
中图法分类号: TP311　　**文献标识码**: A　　**文章编号**: 1001-3695(2004)06-0011-04

Research on Software Architecture Evaluation Methods

HU Hong-lei, WU Guo-qing, LIANG Zheng-ping, LIU Qiu-hua
(School of Computer, Wuhan University, Wuhan Hubei 430072, China)

Abstract: The architecture evaluation of a software system has become more and more important to guarantee the final quality of the system. The purpose of evaluation is to identify the potential risks of the architecture and to verify the quality requirements addressed in the design. This paper presents three mature evaluation methods (SAAM, ATAM, ALPSM) firstly, then compare them in a conceptive framework to find the similarities and differences between these three methods. In the last discusses the combination of multiple methods, the reuse in evaluating the architecture, the introduction of evaluation to design and use in practice.

Key words: Software Architecture Evaluation; Quality Attribute; Scenario; SAAM; ATAM; ALPSM

1　引言

近几年来,软件体系结构(Software Architecture, SA)[1]成为软件工程发展的一个热门方向。随着对软件体系结构研究的深入开展,逐渐形成了以软件系统的体系结构形式化描述、风格、建模、评估、软件产品线以及基于软件体系结构的软件开发过程等为主要研究内容的一个新领域。

对于软件系统来说,所关注的一个主要问题便是质量,尤其对于大规模的复杂软件系统更是这样。软件体系结构对于确保最终系统的质量有重要的意义。对一个系统的体系结构进行评估,是为了在系统被构建之前预测它的质量,并不需要精确的评估结果,通过分析体系结构对于系统质量的主要影响,进而提出改进。因此,软件体系结构评估的目的是分析 SA 潜在的风险,并检验设计中提出的质量需求[2]。

针对体系结构评估这个新的研究领域,许多研究组织在会议和杂志上提出了众多结构化的评估方法,并且对于评估的方法的改进和实践工作仍在进行中。本文主要讨论三种有代表性的方法,它们可以指导评估人员成功地对系统的体系结构进行评估。这三种方法是: 基于场景的体系结构分析方法(SAAM)、体系结构权衡分析方法(ATAM)、体系结构级别上的软件维护预测(ALPSM)。

收稿日期: 2003-05-10; 修返日期: 2003-06-24
基金项目: 国家自然科学基金资助项目(69873035)

2　主要的术语

2.1　软件体系结构

(1) 定义: 软件体系结构定义很多,本文采用为大多数人所接受的一种定义: "软件系统或计算系统的软件体系结构就是系统的一个或多个结构,它包括软件组件,这些组件的外部可见属性以及组件之间的相互关系"[4]。这个定义仅仅关注系统内在的方面,而大多数的分析方法都是基于这个定义的。

这个定义具有如下的含义:

①SA 是一个或多个系统的抽象。SA 以抽象的组件(Component)来表示系统,这些组件具有外部可见属性,并且相互之间是有联系的,这种联系有时被称为连接件(Connector)。

②SA 是一种可重用、可传递的系统抽象,而组件的细节部分不属于体系结构的范畴。

③系统由多个结构组成,通常也称为视图(View)。任何一个视图只能表示 SA 的部分内容,而不是全部。

(2) 描述: 从不同的视角来看体系结构从而构成视图。可以通过不同的视图或是各种视图的重叠得到关于 SA 的有用信息。常见的视图包括: 功能视图(也叫逻辑视图,描述系统功能及其关系的抽象)、并发视图(也叫进程或线程视图,描述系统进程和线程通过数据流、事件及对共享资源的同步进行的交互)、代码视图(它是程序员所看到的视图,组成部分是类、对象、过程、函数及由它们组成的子系统、层或模块)、开发视图(也叫实现视图,描述源代码的结构视图,一般由文件和目

录组成)、物理视图(描述的是软件与硬件的映射和分布关系,也就是从硬件资源的角度来看系统是如何部署的)。

在实际中应该采用哪些视图要根据实际情况来决定,一般应选用能体现 SA 重要信息的视图。Rational 公司的 Philippe Kruchten 提出了一个 4 + 1 的视图模型[5],即由逻辑视图、进程视图、物理视图、开发视图再加上用来解释体系结构描述的用例或场景。

2.2 质量属性

质量属性是一个组件或一个系统的非功能性特征。软件质量在 IEEE 1061[6] 中定义,它体现了软件拥有所期望的属性组合的程度。另一个标准 ISO/IEC Draft 9126-1 定义了一个软件质量模型。依照这个模型,共有六种特征:功能性、可靠性、可用性、有效性、可维护性和可移植性,并且它们被分成子特征,根据各个软件系统外部的可见特征来定义这些属性。

2.3 风险承担者(Stakeholder)

风险承担者就是对体系结构及根据该体系结构开发的系统有自己的要求的人员。风险承担者涉及面很广,可能是最终用户、开发人员、项目经理等。比较特殊的一类人员是项目决策者,即对评估结果感兴趣,并有权作出影响项目未来开发决策的人。体系结构设计师也是很特殊的一位风险承担者,一定要让他参加整个评估过程。

2.4 场景(Scenario)

场景就是对于风险承担者与系统的交互的简短描述。比如用户可能会描述如何使用该系统来完成某项功能,这时场景就很类似于面向对象技术中的用例。在评估过程中,使用场景将那些模糊的不适用于分析的质量属性需求描述转换为具体的易于理解的表述形式。

2.5 评估技术

在体系结构层次上有两类评估技术[4,6]:询问和度量。本文讨论的评估方法都至少采用了这两种技术中的一种,或是两种技术的结合(常同时混合使用,如下文讨论的 ATAM 就是一种混合方法)。

(1) 询问技术。生成一个体系结构将要问到的质量问题,可适用于任何质量属性,并可用于对开发中任何状态的任何部分进行调查。询问技术包括场景、调查表、检查列表。调查表是通用的、可运用于所有软件体系结构的一组问题;而检查列表则是对同属一个领域的多个系统进行评估,积累了大量经验后所得出的一组详细的问题。两种技术都是事先准备好的,由评估人员用于搞清软件开发中反复出现的问题。而场景比起上述两种技术,能更具体地描述问题。

(2) 度量技术。采用某种工具对体系结构进行度量。它主要用于解答具体质量属性的具体问题,并限于特定的软件体系结构,因此与询问技术的广泛适用有所不同。另外,度量技术还要求所评估的软件体系结构已经有了设计或实现的产品,这也与询问技术不同。度量技术通常包括指标(Metrics)、模拟、原型和经验。

3 基于场景的体系结构分析方法(SAAM)

SAAM 于 1983 年提出,是最早形成文档并得到广泛使用的软件体系结构分析方法。它最初用来分析 SA 的可修改性,后来实践证明也可用于其他的质量属性如可移植性、可扩充性等。此方法主要在文献[7]中介绍,该文采用 SAAM 方法对于用户界面软件的体系结构的可修改性进行评估。

SAAM 使用场景作为指定和评估质量属性的表述手段。如果是评估单个软件体系结构,SAAM 将指出体系结构中未能满足质量属性需求的地方,并提出改进的意见。而对于多个体系结构的比较,则明确哪一个能更好地满足质量属性需求。

图 1 描述的是 SAAM 的评估活动。

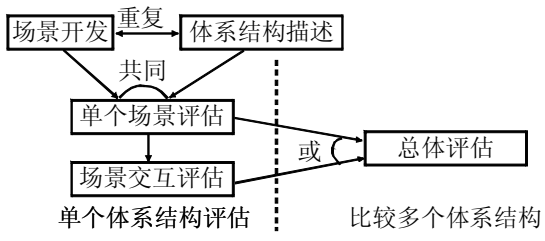

图 1 SAAM 的评估活动

总的来说,SAAM 评估分六个步骤:

(1) 场景开发。通过集体讨论,风险承担者提出反映自己需求的场景。

(2) SA 描述。SAAM 定义了功能性、结构和分配三个视角来描述 SA。功能性指示系统做了些什么,结构由组件和组件间的连接组成,而从功能到结构的分配则描述了域上的功能性是如何在软件结构中实现的[7]。场景的形成与 SA 的描述通常是相互促进的,并且需要重复的进行。

(3) 场景的分类。在分析过程中需要确定一个场景是否需要修改该体系结构。不需要修改的场景称为直接场景,需要修改的场景则称为间接场景。另一方面需要对场景设置优先级,以保证在评估的有限时间内考虑最重要的场景。

(4) 单个场景的评估。主要针对间接场景,列出为支持该场景所需要对体系结构做出的修改,并估计出这些修改的代价。而对于直接场景只需弄清体系结构是如何实现这些场景的。

(5) 场景交互的评估。两个或多个间接场景要求更改体系结构的同一个组件就称为场景交互。对场景交互的评估,能够暴露设计中的功能分配。

(6) 总体评估。按照相对重要性为每个场景及场景交互设置一个权值,根据权值得出总体评价。

SAAM 是一种成熟的方法,已被应用到众多系统中,这些系统包括空中交通管制、嵌入式音频系统、WRCS(修正控制系统)、KWIC[8](根据上下文查找关键词系统)等。

4 体系结构权衡分析方法(ATAM)

ATAM 是在 SAAM 的基础上发展起来的,SAAM 考察的是软件体系结构单独的质量属性,而 ATAM 提供从多个竞争的质量属性方面来理解软件体系结构的方法。使用 ATAM 不仅能看到体系结构对于特定质量目标的满足情况,还能认识到在多个质量目标间权衡的必要性。

在 ATAM 中,体系结构设计师采用五个基本结构来描述 SA,即来源于 Kruchten 的 4 + 1 视图,只是将逻辑视图分成了功能和代码结构,并且加上这些结构之间适当的映射关系。同时根据需要,还要采用其他的视图:动态视图,表明系统如何通信;系统视图,表明软件是如何分配到硬件的;源视图,表明组件和系统如何组成了对象。

ATAM 的评估过程分为九个步骤[9]。图 2 给出了每一个

步骤, 并将其分为四个阶段, 图中循环的箭头体现了 SA 设计和分析改进可能的迭代过程。

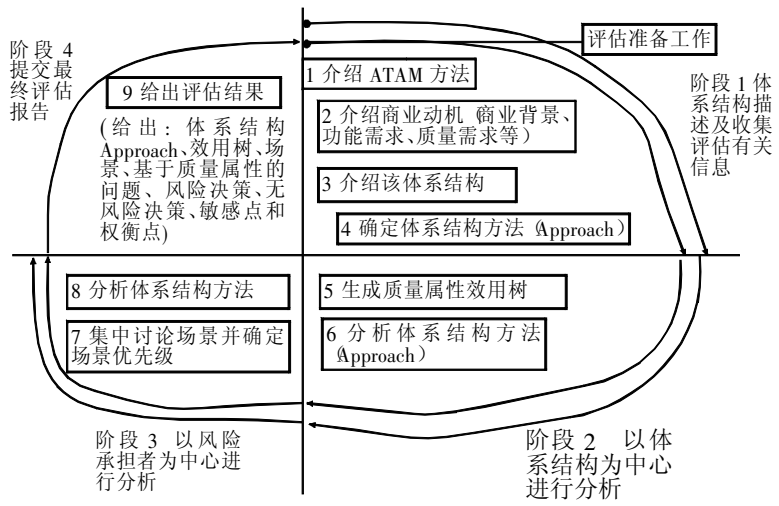

图 2　ATAM 评估的步骤

ATAM 通过理解体系结构方法(Approach) 来分析一个体系结构。体系结构方法是设计师设计过程中采用何种体系结构风格[4] 的决策, 它体现了实现系统的质量属性目标的策略。在 ATAM 中, 基于属性的体系结构风格(Attribute-Based Architectural Style, ABAS) 有助于体系结构风格的概念转换为基于特定质量属性模型的推理。在步骤 4 中就是通过设计师的描述, 从而确定该体系结构为满足质量属性需求所使用的体系结构方法。

通过生成质量属性效用树, 将商业驱动因素以场景的形式转换成具体的质量属性需求, 并对这些质量属性场景设置相对优先级。步骤 6 中将步骤 4 中确定的体系结构方法与生成效用树得到的质量属性需求联系起来进行分析, 从而确认与效用树中最高优先级质量属性相联系的体系结构方法, 生成针对特定质量属性的提问, 并且确认有风险决策、无风险决策、敏感点和权衡点[10]。

风险承担者参与集体讨论场景并设置场景的优先级, 再将这一组场景与效用树中生成的场景进行对比。场景分为三类: 用例场景(系统的典型使用, 被用来获取信息) 、生长场景(预料到的对系统的更改) 和探索场景(期望用来 "压垮 "系统的极端更改) 。使用不同类型的场景, 可以从多个角度来考察体系结构。

最终的评估结果所包含的内容也可以从图 2 上看出来。

在体系结构的设计过程中, ATAM 提出了一种迭代的改进。当所有的评估活动结束后, 将评估的结果与需求作比较。如果系统预期的行为和需求能充分地接近, 设计者就可以继续进行更高级别的设计或实现; 如果分析发现了问题, 就对所设计的 SA、模型或需求进行修改, 从而开始了一次迭代的过程。随着发展, ATAM 愈来愈成熟, 将被用到众多系统中。

5　体系结构级别上的软件维护预测(ALPSM)

ALPSM 方法通过在软件体系级别上考察场景的影响, 来评估一个软件系统的可维护性。可维护性在 IEEE 610 中定义。该方法采用场景来具体化可维护性需求, 并且用来分析体系结构, 对于系统所需的维护性工作做出预测。预测的结果既可用来比较两个可供选择的体系结构, 也可用来平衡可维护性与其他的质量属性。图 3 给出了 ALPSM 评估方法的输入和输出。

ALPSM 方法包括如下六个步骤[11]:

(1) 确认维护任务的分类。这种分类是基于应用或特定域的, 因此并不抽象。

(2) 合成场景。选择对于维护类别有代表性的场景, 一般每一类选 10 个场景。这里的场景与通常讲的描述系统行为的用例场景不同, 它描述的是与系统相关的可能发生的活动或活动的序列。一个变化场景则描述了系统的某个维护任务。

(3) 给每个场景分配一个权值。定义权值为在某个特定间隔时间内, 这个场景导致一个维护任务的相对概率。产生场景的权值要么使用历史维护数据来推断, 要么由体系结构设计师或域专家来估算。

(4) 估算所有组件的大小。组件的大小影响在组件中实现一个改动所需的工作, 因此通过估算组件的大小来估算维护工作。

(5) 分析场景。对于每个场景, 评估在体系结构及其组件中实现该场景带来的影响, 最终发现哪些场景受到影响及被改变到何种程度。

(6) 计算所预计的维护工作。预测值是每个维护场景的工作的平均加权, 体现了每个维护任务的平均工作量。用下式来表示:

$$M_{tot} = \sum_{n=1}^{Ks} P(S_n) \cdot \sum_{m=1}^{Kc} V(S_n \cdot C_m)$$

P(Sn) 场景 n 的概率权值; V(Sn, Cm) 场景 n 中受影响的组件 m 的数量; Ks = 场景的数目; Kc = 体系结构中组件的数目

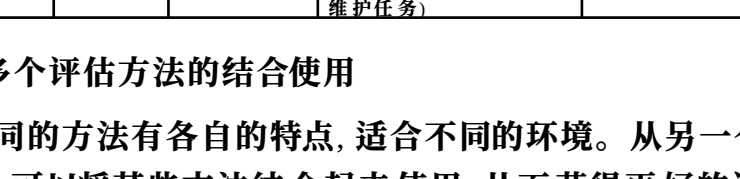

图 3　ALPSM 的输入和输出

ALPSM 方法适用于 SA 设计期间, 而且还能在设计过程中反复地进行评估。它仅仅需要体系结构设计人员参与, 不需其他的风险承担者, 而且还考虑到专家意见和历史数据的使用。此方法已经用在一个血液渗析系统中[11]。

6　讨论

6.1　评估方法的比较

从不同的方法所考查的系统的质量属性、该方法适用的阶段、评估过程中使用的评估技术以及风险承担者的参与情况四个角度定义了一个概念上的框架来比较前面提到的三种方法。通过这个框架可以看出三种不同的评估方法在体系结构评估过程中的相似和差异。表 1 给出了最后的比较结果。

表 1　SAAM, ATAM, ALPSM 三种评估方法的比较

比较因素 / 评估方法	考查的质量属性	适用阶段	使用的评估技术	风险承担者的参与
SAAM	可修改性	SA 的最终版本	场景	所有的风险承担者
ATAM	多个质量属性(侧重可修改性、安全性、性能和可用性)	SA 的最终版本或设计的重复改进过程	询问技术和度量技术 (通过效用树和对场景的集体讨论来理解质量属性需求, 通过对软件体系结构方法的分析来确定敏感点、权衡点和风险)	所有的风险承担者和体系结构设计师
ALPSM	可维护性	设计过程	场景(不同于用例场景, 它描述的是与系统相关的可能发生的活动或活动的序列。而一个变化场景描述了系统的某个维护任务)	仅仅设计师

6.2　多个评估方法的结合使用

不同的方法有各自的特点, 适合不同的环境。从另一个角度来说, 可以将某些方法结合起来使用, 从而获得更好的评估效果。从上面可知, SAAM 方法评估方法仅应用于 SA 的最终

版本,并且需要所有风险承担者的参加;而 ALPSM 方法只需要设计师,并且可以在设计过程中反复使用,耗用较少的资源和时间。对于体系结构评估而言,风险承担者的参与不仅能够促进他们之间的交流,还能加深对系统质量属性的了解。因此,将 SAAM 与 ALPSM 结合起来是一个很好的途径。

不同的评估技术的结合使用也能改进现存的评估方法。现有的评估方法很多是基于场景的,也有基于特定质量属性的,故可以将基于场景的评估技术与质量属性的特定分析技术相结合。ATAM 方法就综合了多种评估技术,因而具有分析的通用性和灵活性,可用于对任何质量属性的评估。

6.3 评估方法中的重用

体系结构的评估中有很多因素可以重用,比如类似系统中经常出现的场景、多次发现的同种类型的风险、质量属性刻画和评估时提的问题等。使用这些可重用的因素,能够提高评估工作的效率。

其中,ATAM 方法使用 ABAS 来实现重用。ABAS 能够为体系结构设计师提供预先封装的分析和提问的集合,这些集合基于对复发问题的解决方案及使用这些方案时的已知困难。在多次的评估过程中,将反复出现的针对质量属性的重要问题和典型的解决方案构建成一个 ABAS,有助于重用与某个重复出现的体系结构方法相关的分析结果,而将反复遇到的某一种体系结构风格或方法(Approach)也构建成一个对应的 ABAS,那么可以重用在分析与该方法相关的质量属性时做的推理。ABAS 也可以使得重用那些反复出现的风险、敏感点和权衡点。

6.4 在体系结构设计中引入评估

ATAM 和 ALPSM 都可以在体系结构设计的迭代过程中使用,从而改进体系结构的设计。而文献[14]中提出一种通过反复的评估和转换来进行体系结构设计的方法。除了场景,还使用模拟、数学建模和客观推理来评估系统的非功能性需求。如果评估的结果不满意,则使用体系结构转换来改进,即利用体系结构风格、体系结构模式、设计模式、转换非功能性需求(NFRs)为功能性的需求和分发 NFRs 来实现转换。转换后的体系结构再次进行上述的评估,从而形成一个反复的设计过程。

6.5 实践中的使用

体系结构的评估是一项实践性非常强的工作,因此应该与实践紧密联系起来。SAAM,ATAM,ALPSM 都已经应用到若干个实际的系统中,并且在实践的过程中不断地得到改进。在很大程度上评估的质量与风险承担者的积极参与分不开,评估的过程同时也是风险承担者对于最终软件系统的认识交流与提高的过程。在具体使用过程中应该针对碰到的情况对这些方法作些调整或修改,但是评估方法的思想、基本的评估技术都应该得到很好的体现。实践中,专家的经验、现有知识库的重用都是值得考虑的因素。

7 结论

纵观本文介绍的三种有代表性的体系结构评估方法,SAAM 以场景为中心,简单易用,但仅仅是粗粒度的分析;而 ATAM 来源于 SAAM、体系结构风格和质量属性,把对多个质量属性的权衡放在首要位置,并结合场景;ALPSM 则是一种轻巧灵活的方法,为设计师提供一个度量工具,使得可以在设计期间重复地评估软件体系结构。SAAM,ATAM 和 ALPSM 都是比较成熟的技术,可以促进对于设计的体系结构的理解,更能提高体系结构设计的质量。

从体系结构评估方法总体的发展来看,一方面,现有的评估方法在潜在风险识别和质量属性预测方面有所进步,但是仍需进一步地求精和改进,然而对于体系结构的描述还没有可广泛应用的体系结构描述语言(ADL),仍然制约着体系结构的评估工作;另一方面,对于质量属性的表述缺乏统一的理解,这些也需要得到进一步地改进。现有的评估工作都是人工来执行,需要专家的参与。若能基于现有的技术,开发一个自动分析工具来帮助设计师评估体系结构,将是很有意义的工作,文献[15]在这方面做了有益的尝试。

参考文献:

[1] avid Garlan, Mary Shaw. An Introduction to Software Architecture [R]. Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.

[2] Li, S Henry. Object-Oriented Metrics that Predict Maintainability [J]. Systems and Software, 1993, 23(2): 111-122.

[3] Buschmann, R Meunier, P Sommerland, et al. Pattern Oriented Software Architectures, a System of Patterns [M]. Chichester: Wiley & Sons Ltd., 1996. 362.

[4] Bass, P Clements, R Kazman. Software Architecture in Practice [M]. Reading, Mass.: Addison-Wesley, 1998.

[5] hilippe Kruchten. Architectural Blueprints: The "4 + 1" View Model of Software Architecture [J]. IEEE Software, 1995, 12(6): 42-50.

[6] Abowd, L Bass, P Clements, et al. ™Recommended Best Industrial Practice for Software Architecture Evaluation [R]. Technical Report, CMU/SEI-96-TR-025, 1997.

[7] Kazman, L Bass, G Abowd, et al. ™SAAM: A Method for Analyzing the Properties of Software Architectures [C]. Proc. 16th Int'l Conf. Software Eng., 1994. 81-90.

[8] L Parnas. On the Criteria to Be Used in Decomposing Systems into Modules [J]. ACM, 1972, 15(12): 1053-1058.

[9] Kazman, M Klein, M Barbacci, et al. ™The Architecture Tradeoff Analysis Method [C]. Proc. Fourth Int'l Conf. Eng. of Complex Computer Systems (ICECCS'98), 1998. 68-78.

[10] Clements, R Kazman, M Klein. Evaluating Software Architecture: Methods and Case Study [M]. Reading, Mass.: Addison-Wesley, 2002.

[11] O Bengtsson, J Bosch. ™Architecture Level Prediction of Software Maintenance [C]. Proc. Third European Conf. Software Maintenance and Reeng., 1999. 139-147.

[12] EEE Std. 610. IEEE Standard Glossary of Software Engineering Terminology [M]. 1990.

[13] Klein, R Kazman, L Bass, et al. ™Attribute-based Architectural Styles [C]. Proc. First Working IFIP Conf. Software Architecture (WICSA 1), 1999. 225-243.

[14] an Bosch, Peter Molin. Software Architecture Design: Evaluation and Transformation [C]. Proceedings of the 1999 IEEE Engineering of Computer Based Systems Symposium (ECBS99), 1999. 4-11.

[15] ohan Muskens, Michel Chaudron, Rob Westgeest. Software Architecture Analysis Tool [C]. Proceedings of the 3d Progress Workshop on Embedded System, 2002.

作者简介:

胡红雷(1978-),男,湖北麻城人,硕士生,主要研究领域为软件工程、中间件技术;毋国庆(1953-),男,教授,博士生导师,主要研究领域为软件理论、需求工程;梁正平(1979-),男,湖南涟源人,硕士生,主要研究领域为软件工程、形式化;刘秋华(1969-),女,湖北黄陂人,硕士生,主要研究领域为软件工程。