# 软件解析学——要点与实践

张冬梅 韩 石 楼建光等
微软研究院

**关键词：**软件解析 软件数据挖掘 技术成果转化

软件生命周期中会产生大量的、各种类型的数据，例如开发过程中的源代码、需求文档、缺陷报告、测试用例；系统运行中的运行日志、性能度量、事件记录；用户交互中使用行为序列、调查问卷、社交媒体反馈等等。随着互联网软件服务的普及，数据的种类更加丰富，规模也越来越大。基于这些数据，软件从业者可以提取出关于软件质量和开发动态的重要信息。因此，数据对于现代软件开发的作用日益明显，并且至关重要。

软件解析学 (software analytics)[1,2] 运用机器学习、数据挖掘、信息可视化以及大规模数据处理等技术，旨在帮助软件从业者以数据驱动的方式进行软件的开发、运行和维护，有效处理、浏览和分析软件生命周期中生成的数据，从中提取有用的信息，做出正确决策。软件解析学关注大数据分析技术在软件行业中的具体应用，是对现代软件工程方法的有效拓展，同时也与软件开发的实践密切相关。

## 软件解析学概述

### 研究领域

软件解析覆盖软件生命周期的各个阶段，涉及相关的不同领域。具体来看，软件解析着重于软件系统、软件用户、软件开发过程三个方向的研究和应用（如图 1 所示），其目标是通过数据驱动的方法从整体上提升软件的质量、用户体验和开发生产效率。

软件系统从规模和复杂度而言形态各异，跨度很大，既包括移动设备的操作系统，又包括由成千上万台服务器组成的网络系统。对于现代软件系统而言，可靠性、性能、安全性等质量指标是决定其成功与否的关键因素。系统的规模和复杂度逐步提高，更大量、更复杂的数据（例如运行时产生的日志信息）也随之生成，并成为监控、分析、理解和提高系统质量的重要依据。

软件用户的体验是影响软件生命力的重要方面。基于运行过程中所收集的实际用户使用软件的行为数据，软件从业者能够更好地理解客户需求，获得提高用户体验的启示，并促进对用户体验的持续改进。

现代软件的开发过程在发展中呈现出与以往不同的特性。比如，开发过程越来越敏捷化，人员之间的分工更加细致，合作也越来越多。通过对软件开发过程中的数据进行分析，软件从业者得以采用有效的机制来促进软件生产力的提高。

### 目标用户

软件解析面向广泛的软件从业者，其目标用户群涵盖软件行业中的各种角色，既包括传统的软件开发工程师和测试工程师，

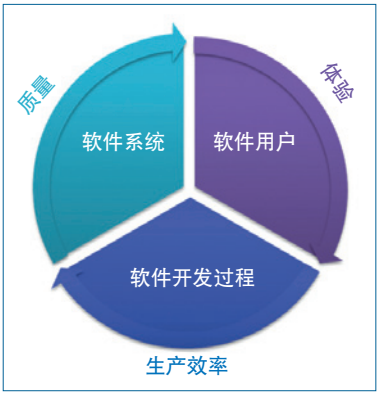



图1 软件解析的研究方向

也包括程序经理、产品经理、运维工程师、用户体验设计师、售前售后支持工程师、数据分析师以及软件项目的管理人员等。不同用户借助于相关的软件解析技术、工具和系统来分析特定数据，完成不同的工作。

## 输出

作为软件解析系统的输出，从软件数据中提取出来的必须是对完成特定软件活动有效的信息。具体来说，输出需要满足以下特征：

- 有揭示性 (insightful)：必须表达出对于完成特定任务来说有意义和有用的知识。这些信息是深层次的，不是通过简单的统计计算就能得到的结果，而是通常需要借助于复杂的分析才能提取出；
- 可操作性 (actionable)：必须能够指导软件从业者很容易地制定出具体的决策或方案来解决相关问题。

如何确保分析结果的有揭示性和可操作性是决定一个软件解析系统在实践中能否成功的关键所在。

## 相关技术

软件解析系统通常需要运用三类技术：大规模数据的存储和计算、机器学习和数据挖掘、数据可视化。

大规模数据的存储和计算适用于有效地管理日益增长的软件数据，并为各种数据分析提供强大的计算平台。机器学习和数据挖掘侧重于各种数据分析的算法，从数据中提取有用信息，支持基于数据的决策分析。数据可视化用于呈现和展示分析的结果，使用户与数据和信息的交互更加方便有效。

设计与实现一个能在实践中应用的软件解析系统是非常具有挑战性的。软件领域中的数据形式、特殊的分析需求以及深入的领域知识，使得现有的数据管理平台、分析算法、可视化设计等往往不能直接应用到软件解析系统中。在解决具体问题的同时，对系统普适性的要求使得设计和实现的难度进一步提高。

## 实践影响

软件解析研究处理的是实际产生的数据，它面向真实的用户和场景，提供可用的系统和工具来解决实际问题，因此它很自然地和软件开发、运行和维护的实践密切相关，并产生较大影响。然而，如何将软件解析的技术和系统成功运用于软件工程的实践，依然存在困难，值得进一步探索。

身处工业界的研究环境使得我们有机会深入接触软件生产的实践，并且帮助我们较早认识到数据驱动的决策方式对于软件实践的重要意义。我们在 2009 年率先提出了“软件解析学”的概念，并成立了软件解析研究组，致力于该领域的前沿研究和实践应用。软件解析学的研究范畴涵盖软件生命周期的各个阶段和方面，强调与软件生产实践的紧密结合，是对软件工程领域相关研究课题 [3, 4] 的升华和扩展。软件解析学的研究逐渐形成了体系，吸引了愈来愈多研究者的关注和参与，现已成为软件工程领域一个活跃的研究分支。

# 软件解析学应用实例

我们通过与不同产品部门的密切合作，在微软公司内部成功部署了多个软件解析系统 [11]。这些系统对于提高相关软件产品的质量、用户体验和开发生产力带来了有效的帮助。

## StackMine——基于大规模运行时数据的操作系统性能分析

性能是软件质量的一个重要组成部分，包括响应速度、任务吞吐率以及资源占用率等，直接影响到用户体验。在各种软件中，操作系统作为基础平台支撑了整个软件生态系统，其性能尤其重要。

### 背景与动机

发现和修复软件的性能缺陷是保证软件质量的重要任务之一。与软件的功能性缺陷相比，性能缺陷的触发条件除了内部和外部的数据状态以外，还包括运行时的时机因素，往往具有更高的复杂性和不可控性。因此，软

件性能缺陷的捕捉、定位以及修复对传统的软件测试和调试技术都提出了挑战。这种挑战对操作系统这类大规模复杂软件则更加明显，具体表现为——在软件产品正式发布前有限的开发和测试周期中，基于有限的硬件、软件、配置和使用场景所组合成的运行时环境，很难捕捉并修复全部的性能缺陷。

为了应对这种挑战，微软公司采用了在软件产品发布后期基于数据分析的策略。首先，微软公司建立了 Event Tracing for Windows(ETW) 平台，在用户实际使用 Windows 操作系统遇到性能问题的时候，能够高效自动地捕捉和收集与该问题有关的系统运行时数据 (ETW trace)。这些数据经过用户的同意会反馈到微软公司。其次，Windows 团队有专门的系统性能分析师对收集回来的 ETW trace 数据进行分析，以发现和定位未知的 Windows 系统性能缺陷，并协调相关的开发团队制定修复方案。

操作系统和 ETW trace 数据的复杂性决定了这项分析任务具有很高的知识和技能门槛。同时，大量有待分析的 ETW trace 数据对系统分析师提出了巨大的挑战。首先，如何基于这些数据建立对 Windows 系统性能缺陷这一问题空间的宏观了解——存在多少不同的性能缺陷？哪些影响范围更大？哪些程度更严重？其次，如何确定分析对象的顺序和优先级以便让影响范围更大、程度更严重的缺陷被尽早地分析？

通过深入研究，我们采用软件解析学手段研发了基于大规模运行时数据的操作系统性能分析系统 StackMine[5] 用于帮助 Windows 系统性能分析师基于大规模 ETW trace 数据进行 Windows 系统性能缺陷的分析（见图 2）。

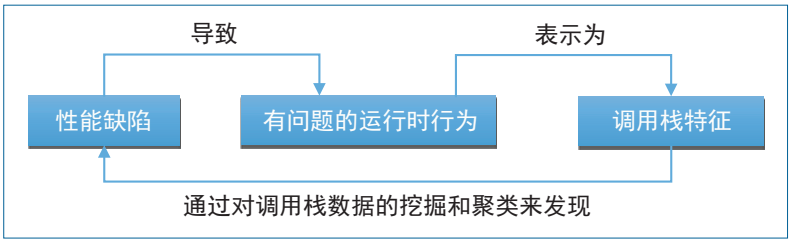



图 2　StackMine问题抽象示意图

## 任务描述

StackMine 的任务是基于大规模 ETW trace 数据，帮助系统性能分析师更高效地发现并分析那些影响巨大、程度严重的 Windows 性能缺陷。这一任务直接来源于微软公司产品部门，属于研究部门与产品部门合作中的需求驱动模式。在一次交流中，一位微软公司产品部门的员工讲述了当时系统性能分析师针对单个 ETW trace 数据文件的分析流程，以及他们面临大规模数据分析时所遇到的挑战。通过进一步的沟通，我们发现这个问题不仅具有重要的实际意义，而且是一个新颖、具有挑战性的研究问题。

## 数据描述

StackMine 采用的输入数据来源于已有平台的支持，即 ETW trace 数据。ETW 的支持从 Windows 2000 开始，并存在于后续 Windows 版本中。简单来讲，每一个 ETW trace 数据文件包含某一次 Windows 性能缺陷发生时间段内重要的系统级事件，比如线程上下文切换、CPU 采样、磁盘读写等等。关于 ETW trace 的详细信息请参见 [6]。

## 解析技术

StackMine 解析技术的研发包括四个方面：

1. 对数据的深刻理解。我们花费了大量时间和精力参阅相关文档并向系统分析师请教，以学习 ETW trace 数据的格式和语义。

2. 对已有实践中重要技术的理解。我们需要学习和研究系统分析师在分析单个 ETW trace 数据文件时的流程，以确定 Windows 性能分析任务中的关键点，即数据的哪些方面是重要的？哪些分析环节是关键？哪些环节存在瓶颈需要改进？

3. 对已有分析算法和技术的定制和改进。StackMine 的核心解析技术是基于数据挖掘领域的经典算法，包括序列数据频繁项挖掘算法和序列数据聚类算法。然而，把经典算法本身简单而直接地应用到实际的分析任务当

中，往往无法得到满意的效果。首先，经典算法需要集成相应的领域知识，使其有针对性地作用于领域相关的分析。StackMine需要把调用栈数据作为序列数据进行聚类，建立调用栈数据的相似度模型成为算法设计的关键。我们需要把系统领域的专家知识结合到序列数据的相似度模型当中，以达到理想的调用栈数据聚类效果。其次，为了有效地处理大规模数据，经典算法需要针对并行计算来进行优化。在StackMine项目中，我们改进了序列数据频繁项挖掘的并行算法，使其能够高效地处理百万级甚至更大规模的调用栈数据。

4. 对复杂的分析结果进行数据可视化呈现。StackMine的分析结果包含大量的聚类，其中每一个单独的聚类代表一个性能缺陷，通常由多个相似的调用栈表示。我们引入了数据可视化技术，用来帮助分析师更加直观和高效地浏览和消化这种复杂的分析结果。比如，我们采用层次图的形式把同一个聚类里的多个调用栈一起展示——调用栈的相同部分合并起来，不同部分会分成不同的分支。分析师可据此直观快速地了解同一个缺陷在多次发生时的共性和变化部分，从而更好地进行缺陷的定位和修复。

### 实践应用

2010年12月，Windows系统分析师应用StackMine进行了一次大规模分析任务，分析对象是从超过6000个ETW trace文件中提取的1.8亿条调用栈数据。经过实践检验，StackMine估计节省了超过90%的用来定位性能缺陷的人力成本。基于Windows性能分析师的判断，在针对1000个ETW trace文件及同等规模的分析任务中，StackMine可以节省4到6周的时间。StackMine在2011年3月完成了从研究项目到生产实践的技术转化，并为Windows 8发现了多个性能可以改进的地方。

## SAS(Service Analysis Studio)——基于数据的在线系统的性能分析

不同于传统套装软件，在线软件系统需要连续运行以保证系统可以提供24×7小时的不间断服务。但是，系统实际运行中时而会发生某些故障，导致服务质量下降甚至服务中断，这种情况通常称为服务事故(service incident)。在过去的几年中，包括亚马逊、谷歌和思杰(Citrix)在内的大公司运作的在线系统都曾经出现过多次服务事故[7,8]。这样的服务事故往往会带来巨大的经济损失，并且可能损害公司的商业形象。正因为如此，在线服务厂商需要投入大量人力、物力来提高对事故的响应速度以及快速恢复系统的能力，从而保证系统的服务质量。

### 背景与动机

事故管理(incident management)[9]的目的是当服务事故发生后尽可能快速地恢复服务系统，保证系统的可用性和质量，维护良好的用户体验。一个典型的事故管理过程通常可以分为事故检测接收和记录、事故分类和升级分发、事故调查诊断、事故的解决和系统恢复等环节。通常，当服务事故发生时，服务监测系统会立即通知相应的软件工程师对事故进行调查、诊断和解决。其中事故调查诊断环节往往是最困难，也是耗时最多的环节。

事故管理的调查诊断环节与数据分析密切相关。这里涉及到的数据是通过监控在线系统收集到的，规模巨大，包括系统运行过程中记录的详细日志(log和trace)，CPU及其他系统部件的性能计数器(performance counter)，服务器、进程和服务程序产生的各种事件(event)等不同来源的数据。这些监测数据往往包含大量能够反映系统运行状态和执行逻辑的信息，因此在绝大多数情况下能够为事故的诊断、分析和解决提供足够的支持。

基于数据对大规模在线服务系统进行高效的事故管理具有很大的挑战性。原因之一是很多在线系统规模庞大，分布在全球多个数据中心的成千上万甚至几十万台服务器构成巨大的分布式系统，每天产生上百亿条日志。在如此海量的数据中查找服务事故的线索无异于大海捞针，费时费力。因此，在实际中需要有自动数据分析技术和工具来帮助软

件从业者进行事故管理，提高事故管理的效率。

在过去的几年中，我们采用软件解析学的方法来解决在线系统中的事故管理问题，开发了一个基于数据的在线系统的性能分析 (service analysis studio, SAS)[10] 系统来帮助软件维护人员和开发人员迅速分析处理海量的系统监控数据，提高事故管理的效率和响应速度。

## 任务描述

我们在设计 SAS 的时候主要面向定位和诊断当前的服务事故，以及快速找到解决事故恢复正常服务的方法。当时设定了四项目标：

1. 对大规模数据进行自动分析，找出与当前服务事故有关的信息，从而帮助工程师快速定位和诊断事故；

2. 对来自不同数据源的不同类型的数据进行分析，并将各个分析结果有效综合；

3. 针对历史上发生过的服务事故，提供一种机制用于积累和利用其相关知识，当一种事故再次发生时，其之前的解决方案可以被直接借鉴，从而大大减少事故解决的时间；

4. 支持交互式的分析 (human-in-the-loop)。事故管理问题本身相当复杂，一个完全自动分析工具通常不能解决所有服务事故问题。因此，人工的介入和交互式分析是非常必要的。在交互中，SAS 应该为工程师提供尽可能准确的相关信息以帮助工程师做出快速准确的推理和决策。

## 数据描述

数据是软件解析的基础，没有高质量的数据就不会产生有意义的分析结果。在 SAS 项目中，我们主要收集和处理了如下几种数据，这些数据都从不同侧面反映了当前系统的执行状态。

- 关键性能指标 (key performance indicators, KPI)。KPI 是用来表征服务质量的指标，例如服务请求的平均延时、失败率等。这些指标能够评价在线服务系统当前的健康状况。
- 系统性能计数器 (performance counters)。性能计数器通常用于记录系统运行时的测量值，包括进程、服务器或者服务本身的资源使用情况和负载情况等信息。
- 系统运行日志。运行日志是一种重要的数据类型，程序员经常用日志信息来调试程序和诊断程序可能的故障。

## 解析技术

针对不同类型的数据和事故管理的两个不同任务，我们研究了多种解析技术并将其应用到 SAS 当中。

1. 可疑信息挖掘

可疑信息挖掘的目的是在大量数据中自动找出可能与当前服务事故相关联的信息，进而帮助定位事故发生的源头和推测事故发生的机理。针对系统中的性能计数器，我们设计了一个新颖的两步方法 [11]。第一步，在所有历史信息中，查找所有具有一定置信度的关联规则作为可能的候选对象；第二步，对每一个候选规则，计算其在当前事故发生时间段的数据上的似然概率比，并将似然概率比高的作为结果输出。针对程序日志序列，我们找到服务事故发生时出错的服务请求，收集它们相对应的日志序列，并将这些出错请求的日志序列与之前系统正常时的日志序列进行比较，找到那些与服务事故非常相关的日志序列片段 [12]。

2. 缺陷组件定位

缺陷组件定位的基本想法是：在大规模的在线系统中，相同的组件往往存在多个同时运行的实例，这些实例应该有类似的行为和表现。如果有个别实例的表现与其他实例很不相同，那么这些表现与众不同的实例（称为“缺陷组件”）是非常可疑的，往往与服务事故相关联。缺陷组件的定位问题可以通过检测野点来实现。这个技术虽然简单，但在实践中非常有效。

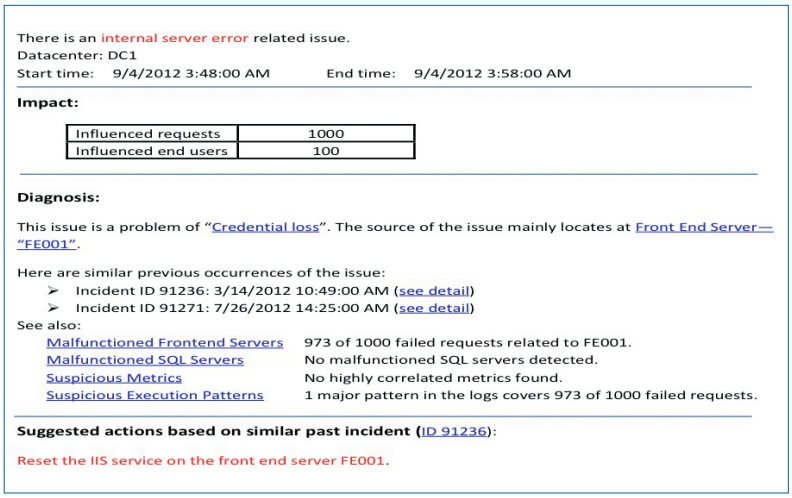
3. 诊断信息重用

当一个服务事故发生后，我们首先自动判断类似的事故在之前是否发生过，如果发生过，则找出之前的相似案例，并在之前相似案例解决方案的基础上，给当前事故提供参考解决方案。这里的关键技术是为每个事故案例

创建指纹 (signature)，并且定义两个案例间的相似度。在 SAS 中，我们利用可疑信息挖掘方法中得到的程序日志片段作为每个案例的指纹，并在此基础上定义了基于推广向量空间模型的相似度度量 [12]。

4. 分析结果综合

为了使 SAS 能够非常容易地被软件工程师们理解和使用，我们将不同分析算法中得到的结果进行综合。根据工程师们进行事故管理的工作流程和场景，我们用一个预先定义的决策树来实现这种综合，并将综合的结果以报表的形式呈现出来( 如图 3 所示 )。报表中提供了工程师们在事故管理实践中最希望得到的信息，包括事故的影响 (impact)、事故诊断信息、系统恢复建议等。基于这些信息，工程师们能够快速做出决策并采取行动。


There is an internal server error related issue.
Datacenter: DC1
Start time: 9/4/2012 3:48:00 AM End time: 9/4/2012 3:58:00 AM

**Impact:**

| | |
|---|---|
| Influenced requests | 1000 |
| Influenced end users | 100 |

**Diagnosis:**

This issue is a problem of "Credential loss". The source of the issue mainly locates at Front End Server—"FE001".

Here are similar previous occurrences of the issue:
- Incident ID 91236: 3/14/2012 10:49:00 AM (see detail)
- Incident ID 91271: 7/26/2012 14:25:00 AM (see detail)

See also:

| | |
|---|---|
| Malfunctioned Frontend Servers | 973 of 1000 failed requests related to FE001. |
| Malfunctioned SQL Servers | No malfunctioned SQL servers detected. |
| Suspicious Metrics | No highly correlated metrics found. |
| Suspicious Execution Patterns | 1 major pattern in the logs covers 973 of 1000 failed requests. |

**Suggested actions based on similar past incident (ID 91236):**

Reset the IIS service on the front end server FE001.

图3 服务事故分析诊断报告样例

### 实践应用

SAS 在 2011 年 6 月被微软公司某在线产品部门采用，部署在全球的各个数据中心，用于一个大规模在线服务产品的事故管理。我们从 2012 年开始收集 SAS 的用户使用记录。通过对半年的使用记录进行分析，我们发现工程师在大约 86% 的服务事故处理中使用了 SAS，并且 SAS 为解决其中大约 76% 的服务事故提供了有效的帮助。

## 技术成果转化的经验总结

软件解析学是一个和软件开发、运行和维护的实践活动密切相关的研究领域。只有将技术成果成功应用到实际软件项目中，才能实现软件解析研究的最终价值。然而，软件解析技术的成果转化是一个复杂的过程。成功的转化受到多种因素的影响，而不仅仅取决于技术本身的优越与否。

### 识别和解决关键的实际问题

开展软件解析通常是从数据开始的。选择一些数据集，运用一些数据分析的技术，试图从中发现一些有意思的结果，不断改进算法以得到更好的结果。然而其所解决的问题往往对软件解析的最终用户（即软件从业者）来说并不重要；或者即使重要，却不是当前最关键的。因此，这样的技术一般只能停留在研究的层面，很难被深入应用到用户的实际工作中。SAS 和 StackMine 在项目的初期都在这个环节上走了弯路。

更有效的做法是以问题驱动的方式开展软件解析的项目。首先要确定一个对于用户当前工作来说非常关键而又足够困难的问题来解决，然后从这一实际问题出发，选择合适的数据集并开发相应的分析技术。这样形成的技术方案直接针对用户亟待解决的关键问题，因而很容易被用户接受并在实践中采用。

如何识别关键的实际问题？一个标准是看它的解决方案能否带来相关任务效益的实质性提高，比如开发生产力、软件质量或用户体验的提升。确定关键问题需要研究人员和用户的紧密合作。用户忙于应对日常工作中的各种挑战，往往不能很好描述要解决的关键问题。此时需要研究人员和用户配合，建立有效的反馈机制，尽早总结、提炼和识别

出关键的实际问题来解决。

例如在SAS项目中，通过与合作的在线服务产品部门的多次交流，我们清晰地确定了他们面对的最关键的实际问题，即显著降低故障后的系统恢复时间(mean-time-to-recover, MTTR)。针对这一目标和相应的挑战，我们从软件解析角度定义了故障管理的问题，结合多种异构数据源，开发了包括一系列技术的完整方案，最后成功部署到生产环境中，取得了明显的成效。

## 提高技术的实用性

为使软件解析技术能真正在实际应用中发挥作用，在设计软件解析的算法和系统时，必须从数据的现状和用户的使用出发，考虑以下目标：

1. 鲁棒性(robustness)：大型软件服务在收集数据时经常出现数据质量的问题，比如数据缺失或噪声。需要精心调整算法的设计以确保其在真实场景下面对各种数据问题足够鲁棒。

2. 高效性(performance)：系统的性能是决定软件解析技术是否会被实际应用的一个重要因素。例如SAS系统的目标是为了加快软件服务故障的调查过程，设计实现该系统时，必须确保分析算法的高效和实时响应。

3. 可伸缩性(scalability)：由于实际问题中的数据规模不同，软件解析的技术方案必须是可伸缩的。我们在StackMine项目中采用了分布式计算的方案，否则无法处理Windows系统调用栈分析的数据规模。

4. 可定制性(customization)：由于各种软件和服务的差异性，软件解析系统必须具备足够的可定制性，以满足不同软件和服务的特定需要。

## 重视系统的开发和部署

在软件解析技术的成果转化过程中，核心算法的研究固然重要，一个可运行的端到端的软件解析的实际系统同样不可缺少。开发和部署这样的系统需要花费研究人员大量额外的精力，但从对成果转化的促进作用来看，这些投入是非常值得的。

我们在每个软件解析项目中，都非常重视实际系统的构建。我们采用分步骤和增量式的方法来逐步增加系统的功能，将工程上的投入控制在可以管理的范围内，同时也从以下三个方面产生实际影响：首先，一个可运行的系统能够展示出研究成果的价值，并且逐步建立起研究人员和软件解析用户之间的信任；其次，一个可运行的系统能够帮助研究人员得到及时的用户反馈；第三，一个可运行的系统能够帮助研究人员建立良好的信誉，并带来更多将来和用户合作研究的机会。

## 理解特定的领域知识

软件领域的数据往往带有该领域特定的语义。只有深入理解了数据的语义，才能有效地进行数据分析。StackMine项目是这方面一个典型的例子。我们必须花费大量时间来弄懂性能数据的语义，否则无法进行有效的分析。

在实践中，对数据的理解包含三方面：数据解释、数据选择和数据过滤。解释数据时，研究人员需要弄清与特定领域有关的基本的定义、术语和概念；选择数据时，研究人员需要清楚数据和所研究问题的联系；过滤数据时，研究人员需要理解数据中的问题和限制，以避免使用不正确的数据。为了理解数据，研究人员和用户之间的沟通和交流尤为重要。研究人员需要通过有效的交互，从用户处获得领域知识和相关的经验技巧，以获得对数据的正确理解。

## 结语与展望

软件解析学是软件工程研究的一个重要分支，它采用基于数据的分析方法来帮助提高软件的产品质量、软件用户的体验以及软件开发的效率。软件解析学与软件的生产实践密切相关，近年来，一些软件解析的研究成果已经被成功地应用到实践当中。随着软件解析应用的不断拓展，软件解析的研究也会在数据收集与管理、分析方法、知识表达、与领域知识相结合以及可视化方面不断深入，并对软件生产的实践产生日益重要的作用。■

**张冬梅**

CCF会员。微软研究院研究员。主要研究方向为软件解析学，数据挖掘，机器学习，数据可视化等。
dongmeiz@microsoft.com

**韩　石**

CCF会员。微软研究院研究员。主要研究方向为软件性能分析、软件解析学、机器学习、数据挖掘等。
shihan@microsoft.com

**楼建光**

CCF会员。微软研究院研究员。主要研究方向为数据挖掘、机器学习等方面的应用。jlou@microsoft.com

**党映农　张海东　谢　涛**

## 参考文献

[1] Dongmei Zhang, Yingnong Dang, Jian Guang, et al.. Software analytics as a learning case in practice:approaches and experiences. *In Proceedings of International Workshop on Machine Learning Technologies in Software Engineering (MALETS 2011)*, 2011, 55~58.

[2] Dongmei Zhang, Shi Han, Yingnong Dang, et al.. Software analytics in practice. *IEEE Software 2013*; 30(5):30~37.

[3] A. E. Hassan, T. Xie. Software intelligence: future of mining software engineering data. *In Proceedings of FSE/SDP Workshop on the Future of Software Engineering Research (FoSER 2010)*, 2010:161~166.

[4] R. P.L. Buse, T. Zimmermann. Information needs for software development analytics. *In* Proceedings of the 34th International Conference on Software Engineering (ICSE 2012), *Software Engineering in Practice Track*, 2012:987~996.

[5] Shi Han, Yingnong Dang, Song Ge, et al.. Performance debugging in the large via mining millions of stack traces.*In Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*, 2012:145~155.

[6] http://msdn.microsoft.com/en-us/library/windows/desktop/bb968803%28v=vs.85%29.aspx.

[7] Amazon's S3 cloud service turns into a puff of smoke. *In InformationWeek NewsFilter*, August, 2008.

[8] J. Nicholas Hoover. Outages force cloud computing users to rethink tactics. *In InformationWeek*, August 16, 2008.

[9] http://en.wikipedia.org/wiki/Incident_management_(ITSM).

[10]Jian-Guang Lou, Qing Wei Lin, Rui Ding, et al.. Software analytics for incident management of online services: an experience report. *In Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013)*, 2013:475~485.

[11]Qiang Fu, Jian-Guang Lou, et al.. Performance issue diagnosis for online service systems. *In Proceedings of the 31st International Symposium on Reliable Distributed Systems (SRDS 2012)*, 2012:273~278.

[12]Rui Ding, Qiang Fu, Jian-Guang Lou, et al.. Healing online service systems via mining historical issue repositories.*In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, 2012:318~321.