基于群体智慧的软件开发间层模型及其架构实现

何炎祥[1,2] 杨建康[2] 鲍海洲[2] 冉亚洛[2] 郭波波[2] 杨建喜[2]
(武汉大学计算机学院 武汉 430072)[1] (武汉大学软件工程国家重点实验室 武汉 430072)[2]

摘 要 减少或者避免重复劳动是应对软件危机的一个重要方法。对软件重用技术进行研究,以避免重复劳动为目标,提出利用群体智慧的间层模型来试图解决软件危机问题。提出的柠檬框架是间层模型的一个实现,具有一定的实用性。同时柠檬框架也是正在进行的一个项目。

关键词 重复劳动,软件重用,群体智慧,间层模型,软件危机,柠檬框架
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.1.040

Interlayer Model for Software Development Based on Collective Intelligence and its Architecture Implementation

HE Yan-xiang[1,2] YANG Jian-kang[2] BAO Hai-zhou[2] RAN Ya-luo[2] GUO Bo-bo[2] YANG Jian-xi[2]
(School of Computer, Wuhan University, Wuhan 430072, China)[1]
(State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)[2]

Abstract Reducing or avoiding duplication of effort is an important way to deal with the software crisis. For the purpose of avoiding duplicate effort and trying to deal with the software crisis, interlayer model which makes use of collective intelligence was proposed after necessary software reuse technologies were researched. Lemon framework the paper presented is one of implementations of interlayer model, which is practical and meanwhile it is also an ongoing project.

Keywords Duplication of effort, Software reuse, Collective intelligence, Interlayer model, Software crisis, Lemon framework

1 引言

过去的50年里,软件带来了很多便利,但是对于软件本身而言,其危机却一直存在。软件危机在20世纪60年代首次被提出,一直到现在,软件制造行业从未摆脱危机的影响。在传统观念中,软件制造的困难在于软件行业的不成熟以及软件的复杂性[2-4]。

软件危机表现在[5]:

①软件开发费用和进度失控。费用超支、进度拖延的情况屡屡发生。有时为了赶进度或压成本不得不采取一些权宜之计,这样又往往严重损害了软件产品的质量。

②软件的可靠性差。尽管耗费了大量的人力物力,而系统的正确性却越来越难以保证,出错率大大增加,由于软件错误而造成的损失十分惊人。

③生产出来的软件难以维护。很多程序缺乏相应的文档资料,程序中的错误难以定位,难以改正,有时改正了已有的错误又会引入新的错误。随着软件的社会拥有量越来越大,维护占用了大量人力、物力和财力。进入80年代以来,尽管软件工程研究与实践取得了可喜的成就,软件技术水平有了长足的进展,但由于软件难以维护,软件生产水平依然远远落后于硬件生产水平的发展速度。

④用户对"已完成"的系统不满意现象经常发生。一方面,许多用户在软件开发的初期不能准确完整地向开发人员表达他们的需求;另一方面,软件开发人员常常在对用户需求还没有正确全面认识的情况下,就急于编写程序。

新的时代,危机并没有消除甚至减少,反而出现了新的表现:

①软件成本在计算机系统总成本中所占的比例居高不下,且逐年上升。由于微电子学技术的进步和硬件生产自动化程度不断提高,硬件成本逐年下降,性能和产量迅速提高。然而软件开发需要大量人力,软件成本随着软件规模和数量的剧增而持续上升。

②软件开发生产率提高的速度远远不能满足计算机应用迅速普及深入的需要,软件产品供不应求的状况使得人类不能充分利用现代计算机硬件所能提供的巨大潜力。

在这篇文章中[6],按照时间将软件危机分成了两个过程,称为危机1.0和危机2.0,不同时期软件危机会出现新的表现。

一直以来,出现过很多解决软件危机问题的方案,但从没有一个比较有效的解决方案。被人们视为解决软件危机比较

到稿日期:2014-01-22 返修日期:2014-03-16 本文受国家自然科学基金重点项目(91118003)资助。
何炎祥(1952-),男,博士,教授,博士生导师,主要研究方向为可信软件、并行分布处理、软件工程和嵌入式系统,E-mail:yxhe@whu.edu.cn;杨建康(1990-),男,硕士生,主要研究方向为软件工程;鲍海洲(1988-),男,硕士生,主要研究方向为软件工程;冉亚洛(1991-),男,硕士生,主要研究方向为软件工程;郭波波(1990-),男,硕士生,主要研究方向为软件工程;杨建喜(1988-),男,硕士生,主要研究方向为软件工程。

好的方法有面向对象、人工智能、自动编程、图形化编程、程序验证等[7]，近些年来，人们最关注的则是软件重用[7]。

软件重用虽然取得了一些进展，但依然存在着影响其前进的障碍。软件重用基本还是应用在个人或者一个集体的内部。不同的人或者集体使用的开发平台不一样，使用的开发语言也会不一样，开发的风格也不能轻松相互理解达成一致，这些使得将重用技术推向整个软件行业极为困难。

将重用技术推向软件行业需要有一个很好的标准，同时这个标准的质量会影响到行业使用这个标准的频率，进而影响到软件重用整体的发展水平，最终撼动到软件危机这个怪物。

既然如此，是否有一种标准可以解决软件危机，促进软件生产的效率，使其有指数级提高呢？Brad Cox 在文献[8]中提到，解决软件危机比起技术而言，人也许是关键因素，软件危机产生的原因是软件的复杂性，重用的目标就是解决重复劳动，降低在构造软件时的复杂程度，但是目前有效的重用也只是在一群集体内部共享，而缺乏有利的诱因来鼓励企业公司个人生产、买卖可重用部件。如果买卖各层部件者皆有利可图并且有一种友善高效的组合规则，那么在重用领域甚至软件领域将是一场变革。

依照 Brad Cox 的理论，关键问题就落在了市场诱因和组合规则二者身上。如果 Brad Cox 的理论设想成为现实，则必然会带动一条价值链，将群体智慧运用其中，从而减少软件开发的复杂性，提高软件的生产效率。在达到一定规模时，必然可以解决软件行业的危机。人们不再是独自或者几个人坐在电脑前各自开发，而是集合世界上所有人的智慧，开发出更为庞大、更为稳定、更为复杂的作品。

基于群体智慧的这种市场诱因和组件规则，建立了一种新型的开发模型，称之为间层模型；同时，基于这个模型开发出了一套框架，我们将它命名为柠檬框架。

2 间层开发模型

间层开发模型是基于群体智慧的一种提供市场诱因和软件复用的开发模型。如图 1 所示，模型将开发分成两条线路：线路 1 和线路 2，线路 1 用来开发复用素材，线路 2 用来开发最终的软件成品；在两条线路上，模型将开发分成了 3 层，两个外层夹着一个中间层，中间层将外层的开发联系起来，同时通过中间层的作用，线路 1 和线路 2 之间产生了内在的关系。

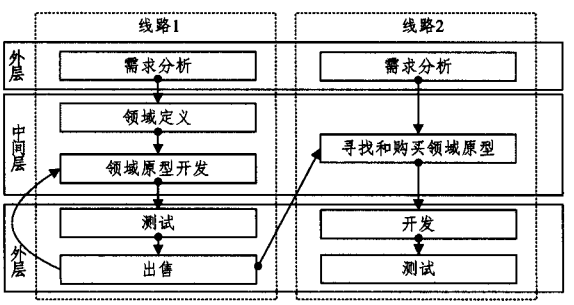

图 1 间层开发模型

图 1 中的领域和领域原型：间层开发模型中领域和领域原型的概念是相互对应的。领域是指面向一定需求的范围，其大小不固定，具体程度也不固定。比如，医学、化学、电子等等是一种比较宽泛的领域，电子的示波器波形噪声分析和提取是一个比较具体的领域。领域原型则是在领域下提供设计开发的套件，这个套件应该包含了所定义的领域中所有的内容，由于领域可大可小，可具体也可抽象，因此领域原型也会出现这种情况，比如一个医学的领域原型可能包含了医学所需要的知识库以及所有的操作，而对于生物样本的录入和统计领域模型可能只需要提供生物的录入接口和统计功能。利用一个领域原型，可以开发出很多个各种各样的软件，它们之间存在着共性，这些共性由领域原型背后的领域决定。领域和领域原型提供了一套高度可重用同时不会降低开发灵活性的规则体系。

线路 1 用来开发领域原型，整体上是流程式的开发，但是其中存在一个迭代：开发出的领域原型可以在领域原型开发时使用。下面依次介绍线路 1 的过程：

1)需求分析，这个阶段需要明白需求分析的对象、方式和目标等等，这与软件工程中需求分析概念一致，但是目标并不是实现软件，而是实现提供领域原型。

2)领域定义，是在需求采集完成之后，确定开发的领域范围以及表现形式等等，这完全都是开发者根据需求和自身的条件进行确定。

3)领域开发，即按照领域定义后的结果进行设计开发和实现，在这个过程中，可以使用传统的开发模型开发，当然与其他开发模型不一样的地方依然是开发的目的，其他模型是为了实现一个具体的软件产品，而这里是实现一个领域原型。在这个阶段可以使用其他的领域原型进行迭代开发。

4)测试，即测试领域设计后的结果的可用性、实用性以及安全性等等。

5)出售，将领域原型出售。领域原型并不是一个完整的软件成品，只提供一个领域内软件开发的基本开发资源。它可以出售给开发领域模型的开发者进行迭代开发，也可以提供给开发软件的开发者开发软件成品。出售这一环节是整个间层开发模型市场诱因的体现和实现。

线路 2 利用领域模型进行开发，它是线路 1 价值的体现，其整体也是一种流程式的开发。下面依次介绍线路 2 的过程：

1)需求分析，首先和传统开发模型一样，依旧是进行需求的分析，确定目标软件开发的需求。

2)寻找和购买领域原型，在需求分析的基础上，找到与需求分析结果一致的领域原型。

3)开发，在完成领域原型寻找和购买后，按照领域原型的规则进行开发实现。

4)测试，在这个过程中与传统的开发模式不同，因为利用领域原型进行开发，其中的领域原型都是经过线路 1 中测试过的，因此其可用、使用、安全都有保证，所以需要测试的仅仅是额外增加的模块以及整体的测试，理论上减少了测试成本。

可以看到，间层模型的层次结构显示外层围绕着中间层，中间层处于核心地位。与中间层相联系的活动有：线路 1 的需求分析、线路 2 的需求分析、测试、开发和出售。其中出售与中间层的两个活动均产生了联系。中间层是一个重要的枢纽，它承接上下，同时又将两条线路连通。

中间层的 3 个活动均与领域相关，中间层的作用是领域作用的体现。在间层模型中，所有的开发都是围绕着领域原型产生的，但是将领域原型与外界其他活动联系起来则是中

间层的贡献。

间层开发模型中存在一条价值链：出售—购买，这个价值链就是一个存在于社会文化中的诱因，推动软件开发中人的因素，同时在这个价值链和开发线路的分隔下，可以促进开发的社会化和全球化，运用群体智慧产生不可思议的成果。领域模型的出现，减少了软件的重复劳动，是对软件组件等重用思想的补充，增加了重用的类型，提高了重用的效率。两条开发线路使得开发过程真正分离，有利于全球化、社会化的生产。总而言之，无论从物还是从人的角度看，间层模型都提供了一种开发的新视角。

间层开发模型虽然提供了一种软件开发的新思想，但是它的实现需要依靠众多开发者的支持，无法存在于一个或者一个集团内部。支持间层开发模型的框架应该提供一个有效的规则将众多开发者组织起来，同时还需要具体实现领域模版的细节。接下来介绍基于间层开发模型的一个框架设计，我们称它为柠檬框架。

3 柠檬框架设计

间层开发模型的两个核心：诱因和友善高效的组合规则。围绕这两点和完善其他细节后，我们设计出柠檬框架。

在柠檬框架中，与间层开发模型的领域原型相对应的概念是骨架。骨架在柠檬框架中是一种用来组织重用部件的一套规则体系，但是它本身也是一种重用部件，任何基于柠檬框架的开发，都需要以骨架作为入口进行开发。骨架在存在形式上类似于模版，都是在内部提供了可重用的资源，但是与模版不同的是，骨架比模版这个概念更为灵活、复杂，骨架可以扩展。骨架可以是一个完整的开发成品，也可能只是提供了工具，骨架的具体实现依赖于骨架的开发者。骨架存在的最大作用在于给重用部件提供了一套友善高效的整合规则。不同的领域对软件存在不同的需求，在柠檬框架中，骨架可以遍及每个领域，而每个骨架又对应着特定的功能，可以通过骨架的重用来满足不同领域对软件的不同需求。

图 2 显示了柠檬框架的整体构造，这个框架图是以流程式的思维绘制的，其中包含了 4 个角色，下面分别对每种角色以及角色之间的关系进行介绍。

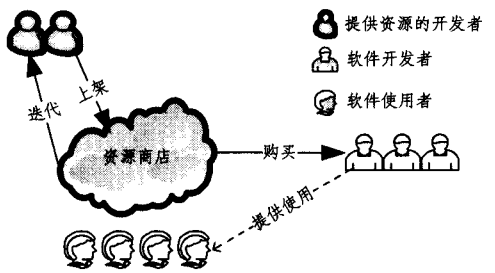

图 2 柠檬框架

1)提供资源的开发者：提供资源的开发者提供开发所需要的重用资源。不同类型的软件需要的重用资源不同。骨架作为一种基本的重用资源，每种软件开发的时候必须使用它并且只能使用一个。除此之外，还需要图片、声音、逻辑等等这些基础的重用资源。提供资源的开发者开发这些资源并且在利益价值的驱使下将开发的资源上架到资源商店，使其面向开发软件的开发者。相比于软件的开发者和软件使用者，提供资源的开发者数量是最少的，他们开发的资源由资源商店进行保存和管理。

2)资源商店：资源商店在整个框架中是一种权利机制以及价值链的核心点。资源商店服务于两端：上端，为提供资源的开发者保存资源，并且还有着利益分成权力，同时还为他们提供了一种迭代开发的环境。作为提供资源的开发者，除了利用一般的方法进行开发外，还可以从资源商店中获取资源进行开发，形成一种闭环迭代的开发路线。下端：为软件开发者提供搜索，提供展示，最终提供购买。

3)软件开发者：软件开发者是具有实际功能软件的开发者，他们可能满足自身需求，也可能满足他人或者市场需求，总之他们开发出来的软件提供给软件的使用者使用。这类开发者的数量在提供资源的开发者和软件使用者之间。

4)软件使用者是整个软件开发的终点，是价值链的落脚点。

角色之间的关系主要是有 4 种操作：上架、购买、迭代和提供。前文已经介绍了这几个操作在角色关系中起到的作用，下面介绍一下这些操作本身的含义。

1)上架：在柠檬框架中，上架指提供资源的开发者签署相应责任书并且制定自己提供的资源权限和定价等信息后上传至商店，经过审核后，在商店合适位置进行展示。因为责任书要求只有提供资源的开发者本人开发的资源或者有权利获得的资源才能在框架体系中流通，所以签署责任书解决了版权问题。

2)购买：对软件开发者而言，为了实现快速开发，他们可以直接从资源市场按照价格获得需要的资源，这个操作也让资源提供者获得了需要的利益；对提供资源的开发者来说，他们购买的目的是利用他人的资源再迭代开发出新的资源。

3)迭代：提供资源的开发者会在开发一个资源时用到其他的资源，比如开发骨架时用到其他的图片、声音等资源，这时就需要从资源市场中获得资源，并且将开发后的资源又放进资源市场，这就是迭代。

4)提供使用：软件开发者会将开发出的具体软件交给使用者使用，这个操作是价值链的终点和落脚点。

如图 3 所示，在整体的框架中存在两条链：价值链和利益链。价值链是开发软件的价值流程，它最终在现实社会中满足各种需求；利益链是开发软件的每个角色获得利益的流程，角色获得的利益除了物质上也有精神上的满足。这两条链主要存在于提供资源的开发者、软件开发者和软件使用者三者之间。

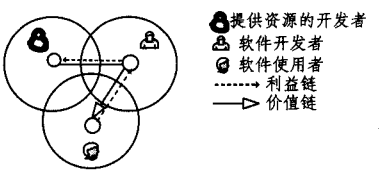

图 3 价值链和利益链图

图 3 中提供资源的开发者、软件开发者和软件使用者之间存在交替的部分，意味着可能存在同时为 2 个或者 3 个身份的人，即同一个提供资源的开发者可能也是软件的开发者，甚至也是软件的使用者。

价值链从提供资源的开发者开始，经过软件开发者，最终落在软件使用者身上，因为开发的最终目的是更好地满足需求，无论是怎样的模型，都需要将价值的终点落在使用者身

上，基于间层模型的柠檬框架也不例外。

利益链有两条，第一条是从软件开发者到提供资源的开发者，软件开发者购买使用资源是其中的体现，另一条利益链是从软件使用者到软件开发者，软件使用者获得并使用了软件并且给予相应的回报，这是这条利益链的体现。这两条利益链是两个诱因，分别促进了提供资源的开发者更好、更积极地为软件开发者提供资源以及促进软件开发者更积极地开发软件。

柠檬框架试着通过利益链和骨架的方式从人的角度解决软件危机。利益链带动人的主动性，产生一种生产和购买的氛围；骨架提供新的重用资源组织形式，它比库、组件等重用形式更为方便、灵活。

同时柠檬框架还将开发者分成了两种：提供资源的开发者和软件开发者。这种分类使得软件开发的任务细分，从而可以在全球范围内开展，达到更好的效率，同时也对软件开发者和软件使用者的关系重新定位。

发布出来的软件安全性和可信性仍然可以采用传统的衡量标准，一样可以经过安全功能测试、渗透测试和验证过程等等，只是由于现有的骨架在上架之前就通过了稳定性和安全性等方面的测试，因此在之后测试的力度上更倾向于整体性能的测试。

4 创新点

间层模型的创新点以及柠檬框架对应的表现为：

1. 两个线路和中间层。将开发过程分割成两个线路，这样分割有利于开发的分工，并且由于两条线路可以并行存在，使得分工得以全球化进行，很大程度上降低了开发的复杂性。中间层使得在分工的过程中又存在着联系，这样在开发软件的整体流程中不会脱节。两条线路在柠檬框架中的表现则是将开发者分成了提供资源的开发者和软件开发者，中间层则是通过开发骨架的活动来实现的。

2. 领域原型。领域原型让软件开发不再是从头开始开发，而是利用领域原型进行二次开发，减少了重复劳动，降低了软件开发不必要的复杂性。这在柠檬框架中对应骨架的概念。领域原型中的领域可大可小，比如开发一个文字游戏领域就比 2D 游戏领域要小，领域可以说是开发者设想开发的软件的一种抽象。

3. 灵活高效的购买。在开发中也许会有图片等可设计可视元素的购买，也会有一整套笨重的源代码的购买，但是像间层模型中将领域原型以及开发的其他要素都作为可购买的实现是一大亮点，这使得重用变得轻松高效。这对应着柠檬框架中的在资源商店购买资源。

5 相关研究

针对软件危机，众多学者研究出了很多方法，还有很多学者正在研究，但是就目前而言，还没有出现特别大的突破。Brooks 认为软件本身的复杂性无法跨越才造成了软件开发越来越困难，而 Brad Cox 认为软件危机源于人，如果可以发挥人的潜力，让人更高效地运作和协作，那么软件危机就可以解决。

目前可以看到，近些年来，软件危机的解决方法都综合了这两位大师的理论。

1）构件模型和构件获取

构件模型（Component Model）是为了研究构件的本质特征和构件之间的关系，是对实际构件的抽象。经过几年的发展，构件模型及其规范已经提出，较有影响的是微软公司的 COM/COM+/DCOM[9]、Oracle 公司的 JavaBean/EJB[10] 以及 OMG 组织的 CORBA[11]。

构件模型为基于构件的软件开发奠定了基础，但是在实际应用构件复用技术时还必须具备大量可供选择的可复用构件以及低成本构件检索技术。构件的获取手段有多种，主要依靠在领域工程[12]的基础上从已有应用系统中挖掘和提炼。

间层开发模型和基于构建开发的区别在于：1）基于构件开发的软件大多采用本原型中的构建实现重用，间层模型的重用方式主要为领域原型的重用，本身而言领域原型的组织结构比构件更复杂，构件实现的是从下至上的组装式开发，而领域原型实现的是从上至下填充式的开发；2）间层开发模型对于相似逻辑的软件可以大规模地重用，从而缩短开发周期，提高软件开发的效率；3）间层模型比起构件更加注重市场诱因机制，在模型中强调了市场的重要性。软件开发者可以实现跨地区合作，根据间层模型开发软件，然后将软件上传到特定的平台，根据开发者开发的软件的实用性和价值给予一定的报酬。

基于构件的开发推广面临的主要困难之一是构件的开发者的设想和组装者的需求并不一定完全匹配，构件组装之后软件的性能很可能非常低下。而间层模型中更加强调市场诱因的作用，因此在开发者设想不完全匹配的情况下可以由此诱因来推动产生；除此之外，由于利用间层模型开发的思想与构建开发的思想有所不同：间层模型的开发思想是由上至下，因此开发者的设想如果与某一个领域原型相符合便可以继续开发，如果不符合只能转至传统的开发模式，从而保证软件的性能。

2）模型驱动技术

模型驱动技术[13]是首先建立模型层次，然后在这个层次上系统自动生成软件的技术。具体来讲，系统首先给出软件设计所需要的功能和非功能特性，这些特性往往比人为设计的效率更高，在功能和非功能的特性划分清楚后，再使用传统的软件设计方法。

3）极限编程/敏捷法

极限编程/敏捷法[14]是近些年来用在快速开发上比较成功的技术，通过减少复杂的开发管理，极大程度上提高软件开发的效率。它是一种以人为核心，反复迭代、循序渐进的开发方法。敏捷法是“适配性”而非“预设性”，即敏捷方法始终认为用户的需求是可变的，依照需求的可能变化来设计软件。敏捷法一出现就受到业界的欢迎，特别适应于那些需求不断变化的信息化系统。

以上的技术和理论一定程度上促进了软件开发的发展，一次次让人们看到软件开发的新大陆。但是，无论什么方法，其对软件危机的撼动只体现出量变，没有一种方法能够大刀阔斧地动摇软件危机问题。

通过群体智慧的力量来完成软件开发，将软件开发的重点转移到人身上，这种方法的关键问题是开发诱因和组合规则。我们的间层开发模型和柠檬框架试图通过解决这两个关键问题，来实现利用群体智慧完成软件开发的设想。

结束语 软件开发依然是一种劳动密集型产业，其开发的复杂度和软件本身的复杂度可以说远远超过了其他行业，

这也是软件危机产生的原因。我们将解决危机的重点放在了人身上，通过群体智慧，实现软件开发社会化和全球化的生产。我们相信利用间层开发模型和柠檬框架将是一个行之有效的方案。

未来，我们将进一步实现间层开发模型的其他框架并完善柠檬框架，最终落实到产品上，以验证间层模型确实是解决软件危机行之有效的方案。

参 考 文 献

[1] Naur P, Randell B. Software Engineering: Report of a conference sponsored by the NATO Science Committee[R]. Germany(Garmisch): Brussels, Scientific Affairs Division, NATO, 1969

[2] Brooks F P. The mythical man-month [M]. Reading: Addison-Wesley, 1975

[3] Brown T. Modernisation or failure? IT development projects in the UK public sector[J]. Financial Accountability & Management, 2001, 17(4): 363-381

[4] Beizer B. Software Is Different[OL]. 2001[2014]. www. soft. com/News/QTN-Online/qtnapr01. html

[5] 钟志永，姚珺. 大学计算机应用基础[M]. 重庆：重庆大学出版社，2012：230-231

[6] Fitzgerald B. Software Crisis 2. 0[J]. Computer, 2012, 45(4): 89-91

[7] Tracz W. Confessions of a used program salesman: institutionalizing software reuse[M]. Addison-Wesley Longman Publishing Co., Inc., 1995

[8] Cox B. " No Silver Bullet" Reconsidered[J]. American Programmer, 1995, 8: 2

[9] What is com? [OL]. http://www. microsoft. com/com/

[10] Enterprise JaveBeans Technology [OL]. http://www. oracle. com/technetwork/java/index-jsp-140203. html

[11] CORBA[OL]. http://www. corba. org/

[12] Kang K C, Cohen S G, Hess J A, et al. Feature-oriented domain analysis (FODA) feasibility study[R]. Carnegie-mellon Univ Pittsburgh Pa Software Engineering Inst, 1990

[13] Zhu L. Model-driven architecture[M]// Mellor S J, Scott K, Uhl A, et al. Advances in Object-Oriented Information Systems. Springer Berlin Heidelberg, 2002: 290-297

[14] Mangalaraj G, Mahapatra R K, Nerur S. Acceptance of software process innovations-the case of extreme programming[J]. European Journal of Information Systems, 2009, 18(4): 344-354

(上接第 158 页)

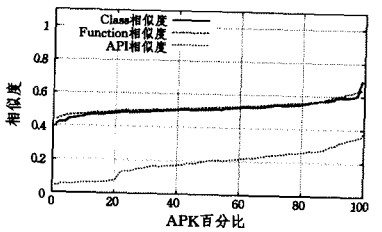

图 6 可信 APK 与恶意 APK 之间的代码比较结果

上面的实验结果分别展示了在 APK 的三层结构中 API 调用情况的比较效果，作为有效的 APK 描述和相似度度量方法，在实际应用中，完全可以将这 3 种比较结合起来，互相弥补在不同情况下的缺陷。例如，将 3 种相似度取平均值或者通过计算 3 种相似度的发散程度，来估计 APK 相似的置信程度。另外，在上面的实验中，我们并未给出具体的可以用来检测 APK 的代码，包括经过了重打包、混淆等过程的恶意代码的有效阈值，而只是展示了在设置合理阈值的前提下三层检测方法的有效性。我们相信有效阈值应该针对不同的恶意类型和检测要求，通过增加预处理和训练的样本数量来动态选择。

结束语 本文提出一种基于静态代码结构的 Android APK 的 API 调用三层描述和相似度比较方法，用以进行恶意 APK 类型和代码族的检测工作，也可用于未知 APK 的风险评估和分类。通过程序实现了整个 APK 处理过程和相似度计算方法。实验数据采用真实多样的已知恶意代码类型和族群，进行了跨类跨族的分层比较方法测试和有效性分析。本文针对以往的 APK 检测特征无法准确描述 API 调用关系和应用范围的缺陷，在 package-class-function 3 个层次上进行了 API 调用情况的统计和对比，增加了代码比较的精确度，放大了恶意代码在实现其恶意行为时无法规避的特定 API 调用需求和分布规律，为 Android 恶意代码特征提取和检测提供了新的视角。在后续研究中，我们将尝试结合目前 3 个层次的相似度，计算出统一的 APK 层面的相似度以及基于恶意相似度的未知 APK 风险程度评估；计划获取并测试更多的恶意代码样本，充分测试文中方法的准确性、适用性和稳定性，尝试针对不同的恶意代码类给出建议的检测阈值。

参 考 文 献

[1] 艾瑞咨询. 2013 年中国移动安全数据报告[EB/OL]. http://report. iresearch. cn/2103. html

[2] 秦中元，徐毓青，梁彪，等. 一种 Android 平台恶意软件静态检测方法[J]. 东南大学学报：自然科学版，2013，43(6)：1162-1167

[3] Canfora G, Mercaldo F, Corrado Aaron Visaggio. A classifier of Malicious Android Applications[C]// Proceedings of 2013 International Conference on Availability, Reliability and Security (ARES 2013). 2013: 607-614

[4] 胡文君，赵双，陶敬，等. 一种针对 Android 平台恶意代码的检测方法及系统实现[J]. 西安交通大学学报，2013，47(10)：37-43

[5] 李寅，范明钰，王光卫，等. 基于反编译的 Android 平台恶意代码静态分析[J]. 计算机系统应用，2012，21(11)：187-189

[6] 杨欢，张玉清，胡予濮，等. 基于多类特征的 Android 应用恶意行为检测系统[J]. 计算机学报，2014，37(1)：15-27

[7] Yang Zhe-min, Yang Min, Zhang Yuan, et al. AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection[C]// Proceedings of the 20th ACM Conference on Computer and Communications Security. 2013

[8] Shabtai A, Kanonov U, Elovici Y, et al. Andromaly: A Behavioral Malware Detection Framework for Android Devices[J]. Journal of Intelligent Information Systems, 2012, 38: 161-190

[9] Isohara T, Takemori K, Kubota A. Kernel-based Behavior Analysis for Android Malware Detection[C]// Proceedings of International Conference on Computational Intelligence and Security (CIS). 2011: 1011-1015

[10] Android ApkTool-A Tool for Reverse Engineering Android APK[EB/OL]. http://code. google. com/p/android-apktool

[11] Dalvik Executable Format [EB/OL]. http://source. android. com/devices/tech/dalvik