北京理工大學

# 硕 士 学 位 论 文 开 题
# 文 献 综 述 报 告

报告题目 **Linux 内核代码分析与可视化技术研究**


学　　号　2120151086

姓　　名　张晶晶

导　　师　刘辉

研究方向　软件理论与工程

二级学科　软件工程理论

一级学科　软件工程

学　　院　计算机学院


2016 年 12 月 8 日

# 文献综述报告要求与打印格式说明

1. 文献综述报告应符合硕士研究生所在学科培养方案的要求。
2. 文献综述报告的内容不再在开题报告中重复，需单独在必修环节中上传。
3. 文献综述报告必须对相关领域已取得之成果进行归纳总结，结合学位论文选题对相关领域未来的发展和研究提出自己的观点。
4. 打印用纸：A4；装订后，学院归档留存备查。
5. 页眉为“北京理工大学硕士学位论文开题文献综述报告”； 加黑宋体，小 3 号，居中。页码居右排版；
6. 页面设计：页眉 2.5cm，页脚 1.5cm，左边距 3cm，右边距 2.4cm，正文用宋体，小 4 号，行间距 26 磅。

# 硕士研究生文献综述报告评阅表

<table>
<tr><td rowspan="4">研究生简况</td><td>姓名</td><td>张晶晶</td><td>性别</td><td>女</td><td>出生年月</td><td>1991 年 9 月</td></tr>
<tr><td>学号</td><td>2120151086</td><td>入学时间</td><td>2015 年 9 月</td><td>身份证号</td><td>14262519910928462X</td></tr>
<tr><td colspan="2">学科、专业</td><td colspan="4">软件工程</td></tr>
<tr><td colspan="2">本科毕业时间</td><td colspan="2">2015.6</td><td>本科毕业学校</td><td>湖北第二师范学院</td></tr>
<tr><td colspan="2">指导小组</td><td>姓名</td><td>职称</td><td colspan="2">工作单位</td><td>签字</td></tr>
<tr><td colspan="2">导师</td><td>刘辉</td><td>教授</td><td colspan="2">北京理工大学</td><td></td></tr>
<tr><td colspan="2" rowspan="4">小组成员</td><td>计卫星</td><td>副教授</td><td colspan="2">北京理工大学</td><td></td></tr>
<tr><td>石峰</td><td>教授</td><td colspan="2">北京理工大学</td><td></td></tr>
<tr><td>王一拙</td><td>讲师</td><td colspan="2">北京理工大学</td><td></td></tr>
<tr><td>高玉金</td><td>讲师</td><td colspan="2">北京理工大学</td><td></td></tr>
<tr><td colspan="3">文献综述报告成绩</td><td colspan="4">□ 优 □ 良 □ 通过 □ 未通过</td></tr>
<tr><td colspan="7">导师评阅意见</td></tr>
<tr><td colspan="7">张晶晶的文献综述报告对大型复杂软件的代码分析与可视化工作进行了总结与分析，并对现有的分析可视化技术进行了详细的阐述。文献综述给出了课题相关的最新进展和研究现状，说明了课题研究的重要意义，为后续研究工作开展奠定了良好的基础。<br><br>签字：<br>年 月 日</td></tr>
</table>

# 1.代码可视化研究现状

软件可视化是指静态的，交互式的或动画的可视化表示的源代码、运行时行为的度量，和进化过程。如 Eclipse 和 Visual Studio 提供了可视化和建模工具帮助程序员理解现有代码。通过绘制统一的建模语言图，如系统架构，组件，类型，相互作用和过程对软件的各个方面进行建模。这些工具大多是独立的，他们在多个窗口和上下文中可视化小规模源代码。内核是由大量复杂的不同类型的依赖关系连接在一起的符号组成的。 可视化工具，可以提高抽象的水平，并减少提交给开发人员的信息量，这样对程序员理解内核将是非常有帮助的。

交互式的内核地图[19]]是一个令人印象深刻的互动网页，显示了内核源代码的复杂的子系统之间的相互联系。这张地图描绘了超过 400 个突出的功能和结构，分为各大子系统。用户可以放大任何功能，并可以移动功能。所有功能之间的关系，通过连接线显示，并点击任何一个功能可以进入到该功能的源代码。交互式的内核是建立在一个图像的基础之上，它是根据内核实现手动创建的。

马里兰大学人机交互实验室的研究人员提出关于 2.5.33 的 Linux 内核的 TreeMap 模型，即用他们的 MillionVis 系统[22] 显示一个正方化的树状图。在他们的设计中，根目录是一个矩形，它被分成几个目录，每个子目录也都是一个矩形。该处理过程是递归重复的，文件和矩形被放置在一个交互式的计算时间内尽可能的平方。每个矩形的大小与该目录下的文件的大小成正比。每个目录的名称都显示在每个顶层目录的矩形的中心，它清楚地显示了不同的大小子系统。他们的 MillionVis 系统只对 Linux 内核源码树做了测试，不能对源码进行可视化。

Linux 内核的图形项目[18]提出一个 Linux 内核后记表示。作者设法对 2.4.0 的 Linux 内核的所有.C 文件产生一个大小为 22MB 的 PostScript 文件。最后一次测试是在 2.2GHZ P4 机器上进行的，大约耗时 20 分钟。

论文[23,27]的作者提出了一个有趣的技术方法，利用调用图的高度可配置的系统的配置复杂度。他们提出用变化的调用图的可视化去捕捉跨配置的依赖关系，并创建可视化，来研究在视觉上通过多样化的影响是如何帮助于开发人员的理解代码的。他们将他们的技术应用到国家漏洞数据库（NVD）中与漏洞相关的 1000 个内核函数。

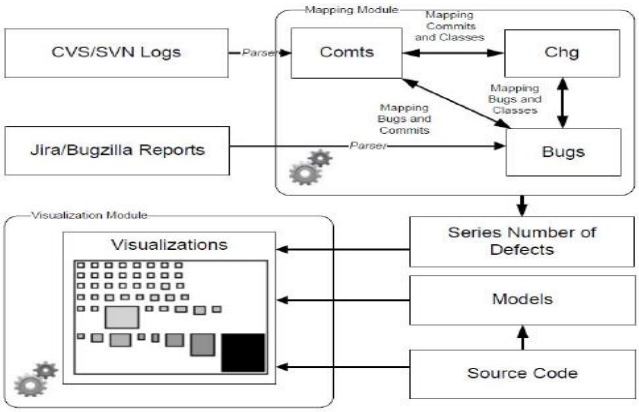
Codesurveyor[11]是一种新的大型空间技术，可以生成的大型软件系统的地图，支持开发者和代码管理者理解代码。它采用了一种类似地图的方式，产生一个可以显示系统抽象结构的互动图，从大陆到各个国家地区，高层层次结构代表大陆，单个源文件和实体的定义表现为各国家和地区。

目前用户使用超链接指向文件很常见，但有时候就会被这种链接关系弄得晕头转向，DeLine[32]最初提出的象征记忆，利用用户的空间记忆来帮助他程序员跟踪他们的代码。DeLine 描述这种技术为软件地形图，将一些软件实体模拟为一个系统，包含的信息有其规模大小和与其他实体的关系。它们之间的距离与他们的亲和力成正比，他们的面积与他们的大小成正比。该算法基于程序的大小将可视化空间成六边形网格，并且对每个方法指定一定数目的连续的六边形网格。

不仅对于代码可视化，在现在的研究中，BUG 可视化可以帮助程序员掌握代码，论文[16]提出了一个工具来支持检索和分析存储在 BUG 跟踪系统的 BUG。该工具从这样的系统中提取的缺陷的时间序列，并将不同的错误的可视化。它的最终目标是促进理解系统。

该工具包括两个模块，映射模块和可视化模块，其架构如图一所示。映射模块主要的功能是：输入两个文件，分别是 logsfiles 和来自 bug 追踪系统的 bug 报告，分别由两个 XML 分析器分析处理 ，一个分析处理器分析 lohgfiles 提取出开发者的评论和改变的 class 文件，另一个分析器处理 bug 报告，收集报告的 bug 和 bug 的标识符，然后把 bug 跟 class 文件关联。该模块将 bug 映射到 class 文件并且创建时间序列。可视化模块的输入是源码、BUGS，输出为可视化结果。可视化模块有两个浏览器，

历史浏览器用来可视化历史 bug，另快照浏览器用来可视化特定的快照系统。可视化部分通过五个示意图展示。




图一：BugMaps 系统架构

Sarita Bassil and Rudolf K. Keller[17]对软件可视化对于程序员理解代码的帮助做了调查，在 2000 年春天对 100 多位参与者做了统计。调查统计结果为当参与者遇到需要分析代码的各种功能，实用性、认知等方面的问题的时候，他们就会求助于软件可视化工具。调查的参与者评价了这些方面的有用性和重要性，并提出了自己平时所需的另外一些方面。参加者一般很乐于使用他们现在用的软件可视化的工具，并且提供了的他们在使用中得到帮助。当然，现有的工具所具备的功能与用户所需的方面也有一些差距。最后，所收集的数据表明，对代码分析方面现有的可视化工具还有很大的研究空间。

# 2.代码解析与可视化

2.1 程序理解原理

T. J. Biggerstaff et al (1993) [1]定义程序理解为：“用不同于源码中的语言解释程序的结构、行为、对操作系统上下文的影响、以及与之相关应用领域的关系”。程序员在阅读理解代码的过程中，首先会通过认知模型，在头脑中形成一个可以表示系统结构的心智模型，然后在心智模型的基础上再返回到源码中去做特定的任务。

程序理解是一个从认知到实践再到认知不断往复的过程，在这个过程中人脑中的心智模型会不断加强。M. P. O’Brien[2]提出一个认知模型由知识库（Knowledge base，Knowledge Graph），心理模型（mental model），外在表示（External representation）和同化过程（Assimilation Process）四个重要要素构成。一般有三种主要模式：Top-down, Bottom-up, Integrated meta-model。V. Mayrhauser et al[4]提出应用这些认知模型可以帮助程序员高效的理解代码。

目前，已经开发了丰富的认知模型，以支持程序的理解。M. P. O’Brien[2]指出，虽然这些模型的重点明显不同，但它们都有四个共同的要素，即知识基础、心理模型、外部表征，和某种形式的同化过程。 M. P. O’Brien[2] 也明确地定义了这些组件。外部可见是协助程序员理解程序的过程中任何“外部”视图，可能是来自软件文档的，或者源代码本身，或来自熟悉该领域的其他程序员的意见，或任何其他类似的源代码代码的观察报告；知识库可以被定义为在他们试图理解代码之前，程序员积累的知识，它会在理解过程中逐渐扩展；同化过程是程序员用来理解源代码的实际策略；心理模型是程序要理解源代码的开发人员的心理表征以后，自己头脑中形成的一个对程序的理解。程序理解通常被称为是程序员理解过程中构建一个软件的适当的心理模型的过程。利用知识库、心理模型，外部表征，同化过程可以不断建立和更新程序员的心理模型。

心理模型的建立和更新，在同化过程中需要使用认知模型。本课题中我们研究三种认知模型，自上而下 （Top-down），自下而上（Bottom-up），集成元模型（Integrated meta-model）。

1984 年， Soloway and Ehrlich [5]提出了一个自上而下的模型，并观察他们的研究，认为一个自上而下的方式是适合熟悉源代码或者源代码的类型的程序员。从理论上讲，如果程序员已经掌握了执行相同的任务和结构完全相同的方式的代码，那么

如果采用自顶向下的模式可以完全了解一个新的程序。A. V. Mayrhauser[4]等人定义的自上而下的模式是以目标为导向，在心理模式上会形成一个层次的目标。

程序理解的自底向上的理论是假设程序员先阅读代码语句，然后将这些语句块转换成更高层次的抽象。这些抽象（块）进一步聚集直到项目的高级认识。Pennington[6]表明，程序员应该理解程序的过程中至少需要两种心智模型。

Von Mayrhauser and Vans 观察到程序的理解的过程既不是简单的自上而下也不是自下而上的过程。A. V. Mayhauser 和 A. M. Vans（1995）开发了一个多层次的理论，这被称为集成模型。这种集成的元模型是从实验的 Von Mayrhauser and Vans 进行，得出的结论是，程序员一般理解程序使用集

2.2 代码解析

GCC，cflow 和 clang 等这些工具都可以分析 C 程序设计语言中的一个源文件，并输出函数之间的调用关系图。Doxygen[8]可以用来解析 C 语言源代码，但它还支持其他流行的编程语言。Doxygen 可以在内核代码的预处理过程中，从为每个源文件中提取代码结构和输出一个 XML 文件。

2.3 代码可视化

在大型复杂的系统中，代码的行数达到千万量级，对于程序员来说理解和维护越来越困难。大的复杂的系统是典型的开发人员几十年来协同工作的产品，这些产品会有一些功能已经过时，但往往文档也不完整，那源代码就是系统行为的唯一可靠的指南，可由于代码量的庞大，使程序员对系统形成一个完整的理解是不可能的。因此在实践中，对于开发人员或者开发团队，系统代码的详细信息是有限的。在维护任务中开发人员由于一个无意识的错误可能会在程序中引起出乎意料的改变，或者在源代码级调用似乎无害的函数或类型就会引起违反机制。这一问题的关键是，目前的提供给开发人员理解大型代码的工具很少。一般都是提供一个 40-80 行窗口进行代码文本阅读，每个文件与整个系统的关系是很难清楚的。大多数提供了一个可折叠的树视图来

展现系统的组织层次结构，但即使这样，对于看大型复杂系统也是有很限的，程序员很容易迷失方向。尤其对于从第三方购买软件的人员，软件的理解对他们更加困难，所以对于技术人员而言，他们更加需要一些策略来为他们的理解代码提供帮助。

可视化技术对于重大科学的发现起到重要作用，为科学新发现创造新的手段和条件。可视化是将数据转换为图形或者图像在屏幕上显示出来，并进行交互处理。可视化图表可以用比文字块 10 倍的速度将陌生的读者带进一个新的领域，借助图形化的手段，清晰有效的传达与沟通信息正是数据可视化的本质。数据可视化技术用形象直观的方式展示结果，使得人们能快速发现数据中蕴含的规律特征。人类的创造性不仅取决于逻辑思维，还与形象思维密切相关。人类利用形象思维将数据映射为形象视觉符号，从中发现规律，进而获得科学发现。大量的实验证明，适当的可视化技术有助于程序员理解代码，可以显著提高程序员的工作效率。

## 5. 参考文献

[1] Biggerstaff, T. J., Mitbander, B. G., Webster, D., 1993. The Concept Assignment Problem in Program Understanding. Proc. International Conference on Software Engineering.

[2] O'Brien, M. P., 2003. Software Comprehension – A Review & Research Direction. Technical Report, University of Limerick.

[3] Peter Chen, Xiaolei Du Improving software comprehension process by Adoption of Cognitive Theories in large-scale complex software maintenance.

[4] Mayhauser, A. V., Vans, A. M., 1995. Program Comprehension During Software Maintenance and Evolution. IEEE Computer, pp. 44-55.

[5] Letovsky, S., Soloway, E., 1986. Delocalized plans and program comprehension. IEEE Software, 19(3), pp. 41 - 48.

[6] Pennington, N., 1987. Comprehension Strategies in Programming. Proc. Second Workshop Empirical Studies of Programmers, Ablex Publishing, pp. 100-112.

[7] Echarts. http://echarts.baidu.com.cn/, 2016.12

[8] Doxygen. http://www.doxygen.org/,2016.

[9] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. Eprint Arxiv, 2013.

[10]J. Hershberger, M. Maxel, and S. Suri. Finding the k shortest simple paths: A new lgorithm and its implementation. ACM Trans. Algorithms, 3(4),Nov. 2007.

[11]Nathan Hawes, Stuart Marshall, Stuart Marshall CodeSurveyor: Mapping Large-Scale Software to Aid in Code Comprehension.

[12] Ahmad Jbara, Dror G. Feitelson On the Effect of Code Regularity on Comprehension

[13] Ahmad Jbara, Dror G. Feitelson JCSD: Visual Support for Understanding Code Controln Structure.

[14]Javier Belmonte,Philippe DugerdilA, Ashish AgrawalThree-Layer Model of Source Code Comprehension.

[15] Dimitar Asenov, Otmar Hilliges, Peter M¨ uller The Effect of Richer Visualizations on Code Comprehension

[16]Andre Hora, Nicolas Anquetil, Cesar Couto, Marco Tulio Valente, Julio Martins BugMaps: A Tool for the Visual Exploration and Analysis of Bugs.

[17] Sarita Bassi1 and Rudolf K. Keller Software Visualization Tools: Survey and Analysis.

[18] Free code graphing project. http://fcgp.sourceforge.net/, April 2016.

[19]The interactive map linux kernel. http://www.makelinux.com/kernel_map/intro,April 2016

[20]Linux kernel archives. https://www.kernel.org/, 2016.

[21] T. Ball and S. G. Eick. Software visualization in the large. Computer, 29(4):33–43, Apr. 1996

[22]J. D. Fekete and C. Plaisant. Interactive information visualization of a million items. In Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02), pages 117–124. IEEE Computer Society,2002.

[23] G. Ferreira, C. K¨astner, J. Pfeffer, and S. Apel.Characterizing complexity of highly-configurable systems with variational call graphs: Analyzing configuration options interactions complexity in function calls. In Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, HotSoS '15, pages 1–2. ACM, 2015.

[24]A. Karahasanovi´c, A. K. Levine, and R. Thomas.Comprehension strategies and difficulties in maintaining object-oriented systems: An explorative study. J. Syst. Softw., 80(9):1541 – 1559, Sept. 2007.

[25]C. K¨astner, P. G. Giarrusso, T. Rendel, S. Erdweg,K. Ostermann, and T. Berger. Variability-aware parsing in the presence of lexical macros and conditional compilation. In Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA ' 11, pages 805 – 824. ACM, 2011.

[26]T. Klemola and J. Rilling. Modeling comprehension processes in software development. In Proc. Int' l COnf. on Cognitive Informatics, pages 329 – 336, 2002.

[27]M. M. Malik, J. Pfeffer, G. Ferreira, and C. K¨astner.Visualizing the variational callgraph of the linux kernel:An approach for reasoning about dependencies[poster]. In Proceedings of the Symposium and Bootcamp on the Science of Security, HotSos ’16, pages 93–94. ACM, 2016.

[28]A. Milanova, A. Rountev, and B. G. Ryder. Precise call graphs for c programs with function pointers. Automated Software Engg., 11(1):7–26, Jan. 2004

[29]K. Mohanta and B. P. Poddar. Comprehensive study on computational methods for k-shortest paths problem. International Journal of Computer Applications,40(14):22–26, 2012.

[30]A. Saebjoernsen, L. Jiang, D. Quinlan, and Z. Su .Static validation of c preprocessor macros. In Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, ASE ’09, pages 149–160, 2009

[31] M. A. D. Storey, K.Wong, and H. A. Mller. How do program understanding tools affect how programmers understand programs. In Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE ’97), pages 183–207. IEEE Computer Society, 1997.

[32]R. Deline, “Staying oriented with software terrain maps,” in Proceedings of the Workshop on Visual Languages and Computation, 2005.