北京理工大學

硕士学位论文
开题报告

选题名称 **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

填 表 说 明

1．只有学籍状态为注册或悬置的研究生才允许开题。但学籍状态为悬置的研究生只有在完成注册手续之后，开题报告及其评审结果才能被认可。

2．硕士学位论文开题报告封面及一至八项必须用计算机输入、打印。

3．开题报告为 A4 大小，于左侧装订成册。硕士研究生应逐项认真填写，各栏空格不够时请自行加页。

4．开题报告经指导教师审阅通过后，由硕士研究生在学科组或更大范围内宣读，并接受专家组质疑、评议。专家组由三名以上高级职称专家组成。开题报告应由硕士生导师为主体组成的评审小组评审。**评审合格后，装订，学院归档留存备查。**

5．硕士研究生应在选题前阅读相关领域的中外文资料，并写出不少于 4000 字的文献综述报告，引用参考文献的篇数不得低于本学科专业培养方案的规定。文献综述报告应反映国际和国内本领域的研究历史、现状和发展趋势。文献综述报告是开题报告的必要附件，开题报告通过后，由学院留存。

6．“参考文献”著录按照 GB7714-87 文参考文献著录规则执行。书写顺序为：序号·作者·论文名或著作名·杂志或会议名·卷号、期号或会议地点·出版社·页号·年。

7. 本件只接受非涉密内容

一 简表

<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>1991.9</td></tr>
<tr><td>学号</td><td>2120151086</td><td>入学时间</td><td>2015.9</td><td>身份证号</td><td>14262519910928462X</td></tr>
<tr><td>学科、专业</td><td colspan="5">软件工程</td></tr>
<tr><td>本科毕业时间</td><td colspan="2">2015.6</td><td>本科毕业学校</td><td colspan="2">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>石峰</td><td>教授</td><td colspan="2">北京理工大学</td><td></td></tr>
<tr><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>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 rowspan="6"></td><td rowspan="2">名称</td><td>中文</td><td colspan="5">Linux 内核代码分析与可视化技术研究</td></tr>
<tr><td>英文</td><td colspan="5">Analysis and Visualization of Linux Kernel</td></tr>
<tr><td>类别</td><td colspan="6">国家项目（ ）： 部（省）项目（ ）； 企业项目（ ）；
自拟项目（ √ ）； 是否兵器类项目（ ）；</td></tr>
<tr><td>性质</td><td colspan="6">基础研究（ ）； 应用基础研究（ √ ）； 应用技术研究（ ）</td></tr>
<tr><td>与导师课题研究课题的关系</td><td colspan="6">是导师研究课题的一部分（ √ ）
与导师研究课题无关（ ）</td></tr>
<tr><td colspan="7">摘 要</td></tr>
<tr><td colspan="7">程序理解是软件维护中的一项重要任务，在软件维护中发挥着日益重要的作用，而随着软件系统规模和复杂性的增大，程序理解也越来费时费力。Linux内核也是一个庞大复杂的系统，代码量已经达到两千万行，对代码理解造成了极大的困难。大量的实验证明可视化技术有助于程序员理解代码，本课题提出了代码分析与可视化技术，通过解析Linux内核源码，并分析数据，再将数据可视化，为程序员提供一个视图界面，帮助程序员快速建立心智模型，提高其理解代码的效率。</td></tr>
<tr><td rowspan="3">关键词</td><td colspan="6">1.关键词限3~5个；2.关键词之间用“；”分隔。</td></tr>
<tr><td colspan="2">中文</td><td colspan="4">代码可视化；Linux内核；程序理解；</td></tr>
<tr><td colspan="2">英文</td><td colspan="4">Source code visualization; Linux Kernel; Program comprehension</td></tr>
</table>

二 选题依据

简述该选题的研究意义、国内外研究概况和发展趋势。

1. 研究意义

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。

在大型复杂的系统中，代码行数已达到数千万行，对于程序员来说理解和维护系统越来越困难。而大型复杂系统往往是开发人员几十年来协同工作的结果，一般源代码是系统行为的唯一可靠途径，由于代码量的庞大，程序员很难对系统形成一个完整的认识。因此在实践中，开发人员一个无意识的错误可能会在程序中引起出乎意料的改变。这一问题的关键是，目前的帮助开发人员理解大型代码的工具一般都是提供一个 40-80 行窗口进行代码文本阅读，程序员一般不清楚每个文件与整个系统的关系。即使有些工具提供一个可折叠的树视图来展现系统的组织层次结构，程序员在理解大型复杂系统时还是很容易迷失方向。

而 Linux 内核代码更是相当的庞大复杂，代码量现在已经达到 2000 万行，程序员理解代码也越来越困难。对于 Linux 内核代码的开发和维护人员，理解代码又是一个必不可少的部分，并且在很大程度上决定工作质量的高低，所以随着代码量和复杂度的增加，理解代码的过程消耗了程序员越来越多的时间和精力。因此，为了提高程序员理解代码的效率，将代码可视化显得更加需要。而在可视化之前需要先分析代码，得到可视化所需的数据。

Linux 内核源码的规模很大并且包含汇编程序，使得分析提取函数节点，以及获得函数之间关系的需要做更多的研究。除此之外 Linux 内核代码中条件编译指令也让分析代码的工作更加艰难，根据不同的编译环境在预处理器中，条件编译指令包含不同的执行路径，而在静态分析中要得到函数的所有执行路径是一个很大的挑战。

2. 国内外研究概况

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。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。

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。

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。

对于函数调用关系的提取，ctags 是一个用于从程序源代码树产生索引文件（或 tag 文件），从而便于文本编辑器来实现快速定位的实用工具。在产生的 tag 文件中，每一个 tag 的入口指向了一个编程语言的对象。这个对象可以是变量定义、函数、类或其他的物件。但是此种方法的缺点也很明显，就是遇到源码量很大的项目时，不能非常准确地给出调用关系。

cscope 是一个 C 语言的浏览工具，通过这个工具可以很方便地找到某个函数或变量的定义位置、被调用的位置等信息

3. 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.

三 研究内容

1. 程序理解的认知过程分析

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。

应用认知模型是提高软件理解的很好办法，但一个完美的可以解决所有问题的认知模型是不存在的，所以选择一个合适的认知模型是很重要的。本课题研究的第一阶段，我们首先会了解为完成不同的任务，程序员应选择何种认知模型完成程序的抽象。

2. 静态分析源代码

现有的工具，例如 GCC、cflow、clang 都可以分析 C 语言源码文件，并且可以输出函数调用关系图。但我们认为代码可视化不仅仅是简单的显示函数之间的调用关系,变量类型和变量之间的关系、BUG 信息、以及版本演化的信息也应该被可视化出来。

因此本课题研究的第二阶段，我们主要选择合适的工具对 Linux 内核代码进行解析，获取 BUG 信息，以及版本之间的变化为第三阶段的可视化做准备。

3. 可视化

在本课题研究的第三阶段，我们主要是选取合适的可视化方式将源码的信息显示出来，给用户提供一个简洁直观的可视化界面，使程序员快速建立心理模型，然后从整体理解代码。

3.1 代码可视化

可视化是为了把大量复杂的数据用一种简洁直观的方式展示出来。如果用现实社会中的某种概念结构可能会使用户更能容易理解。我们可视化的设计会结合人脑理解代码的认知模型，分别从不同的角度可视化代码，对程序员理解代码提供帮助。

3.2 BUG 标注

程序的代码中的 bug 也越来越受到程序员的关注，因为在这些 bug 信息也可以帮助程序员掌握代码，比如可以得知程序每个模块的 BUG 数量，从而得知的那些属于高风险模块，甚至这些 bug 可能会对程序员有一些警示作用，提供一个有效的指导作用。

3.3 版本演化

Linux 内核到现在为止已经到 4.8 版本，从上一版到这一版的更新，总会有一些变化，最基本比如函数的增加、删除、修改，程序员在可视化界面上如果能够清晰的看到版本之间函数的变化，以及不同版本中修改的文件或者函数，对于版本功能的变化可以很快的定位到函数层级。

四 研究方案

拟采用的研究方法、技术路线、实验方案及可行性分析。

1.研究方法

本课题的研究主要分为三个阶段：程序理解认知过程分析、解析分析源码、可视化。 其中程序理解与认知模型的选择密不可分，在理解代码中选择不同的认知模型会形成不同的心智模型，因此选择那些心智模型来代替其人脑对代码的加工过程直接形成人脑对应的内在表现，再将其呈现给用户是我们研究的主要问题，所以第一阶段的工作直接为后续第三阶段工作奠定理论基础；第二阶段主要任务是解析 Linux 内核源代码，用过现有的解析源码工具解析源代码，再设计实验将其分析，得到我们所需要的信息，存入数据库。这一阶段还需要从不同网站获取 Linux 内核 bug 信息存入数据库。第三阶段利用第二阶段分析统计数据，将函数与函数、函数与变量、变量与变量类型之间的关系、以及 bug 数量和版本演化以可视化的方式展示。

2.技术路线

2.1 程序理解的认知过程分析

程序理解原理的分析属于本课题研究的前期工作，我们需要明白人脑理解程序的过程，为第三阶段可视化展示为准备。除此之外还需要清楚认知模型，以及每种认知模型呈现数据的角度。

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

成模型。他们发现在实验中，一般理解大型复杂系统都需要使用这种组合方法。

2.2 静态分析源代码

为了获得快速响应视觉导航，我们需要从内核源中提取有用的信息代码，并存储在数据库中。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。可在我们的可视化希望给用户提供更多的信息，比如函数与全局变量、变量类型和全局变量之间的关系，而不仅仅是简单地显示调用图，所以我们需要更加强大的工具来解析源码。

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。这些 XML 是源码的结构化数据。然后通过实验提取 XML 中关于源码的函数、全局变量以及他们之间关系存入数据库。

2.3 可视化

2.3.1 代码可视化

在 2016 年 10 月发布的 4.8 版本的 Linux 内核，已发展到约 2000 万行代码。毫无疑问，内核是一个庞大复杂的系统，我们的可视化设计应该遵循的人脑的认知模型。为了加强用户的认知过程，我们采用现实社会中的某种概念结构--地图，我们的工具是为用户提供一种像使用地图一样的体验。

地图导航的主要功能是呈现用户给出的一个特定区域的地图，突出的各个区域对象之间的关系、主要道路的详细情况、以及其他一些信息。它还提供其他功能，如旅行路线计算，计算实时交通从一个地方到另一个地方的距离。虽然地图系统和软件系统是两个完全不同的领域，但值得注意的是他们之间有一些相似之处。Linux 内核开发者通常注意内核中的一个子系统。他们关注一个函数的上下文，定位函数的被调函数和调用它的函数，有时候希望能找出两个函数之间是否有一个条调用关系。地图导航和代码导航之间的相似之处如表 1 所列。

地图导航	**代码导航**
本地搜索	函数或者变量搜索
路径规划	调用关系搜索
交通标志	Bug 标注

表 1：地图导航和代码导航之间的相似之处

基于这一观察，我们的工具可以提供如下一系列功能，使得用户可以浏览内核代码。

1) 符号查询：为所有函数、全局变量、变量类型提供一个单一的查询入口。查询的结果为跟搜索符号相关的一系列结果，并且会用力导图显示他们之间的关系。符号查询覆盖的是广泛的，搜索结果可能是与节点相关的成千上万结果。

2) 连接查询：寻找两个可能的相关符号之间的路径，可能是函数与函数，也可能是函数和变量。

3) 结构查询：引用认知模型的自上而下的理论，提供一个结构视图，显示代码库的原始组织，给用户提供一个源码分层的心理模型。用户可访问其任意一个子树，并且可以查询子树中所有的符号以及它们的关系。

4) 源代码查询：检索源代码文本是与文件或符号相关的。引用认知模型的自底向上的理论，提供一个供用户查看相关源代码的功能，有时用户可能想对函数功能做一个深入的分析，就可以查看相关函数的源码。

在连接查询中，寻找两个功能之间的所有路径是至关重要的，但是由于时间复杂度的关系，要查询所有简单路径是不可能。所以在搜索路径的时候我们打算用 k 最短路径路由算法[10]。

在结构查询中，为了在嵌套图中显示他们之间的关系，我们需要把这个大的稀疏的图形切割成更小的组件，使得在最低层的组将中节点和关系边可以显示。通常情况下，图划分是一个很难的问题，一般使用近似算法。论文[9]显示了在图分割的最新进展。在调用子系统图中我们需要应用图分割算法，减少了部分函数节点和边。

我们可视化用的工具是 ECharts[7]，这是一个免费的、功能强大的图表和可视化图库。它可以通过简单的方法给网页中添加互动、高度可定制的图表。它是用纯 JavaScript，是一个全新的轻量级的图库。

2.3.2 BUG 标注

程序的代码中的 bug 类比地图导航中的交通标志，因为 bug 标注可以让程序员得知的那些属于高风险模块，甚至这些 bug 可能会对程序员有一些警示作用，提供一个有效的指导作用。在得到源码的 bug 数据之后，我们可以利用 Echarts 图的特性，在每个模块上添加一个浮动的柱状图，显示该模块的 bug 的数量。

2.3.3 版本演化

版本演化在界面显示中，通过利用 Echats 图中力导图的特性，可以修改图中节点和边的颜色，通过用颜色区分，使得用户可以清楚的看到版本之间函数或者变量的增减，使得对于版本功能的变化可以很快的定位到函数层级。

3.实验方案

在本课题研究中，我们首先了解对人脑理解程序的机制，了解认知模型的对程序理解的作用。接着用合适的工具解析最新版的 Linux 内核源码，得源码的 XML 文件，在设计一系列实验，分析 XML 文件，从 XML 文件中获取所需信息，包括函数、变量、变量类型、函数之间的调用关系等等。最后通过用 Echats 图将源码可视化到网页。

在实现可视化的过程中，包括可视化整个源码的层级结构、函数之间的关系、以及 bug 标注和版本演化。可视化源码层级关系用图划分算法；在路径搜索中，由于无法显示所有的函数之间的路径关系，所以前 k 条用最短路径算法，得到部分调用路径；Bug 标注用柱状图显示 bug 数量的多少。

4.可行性分析

4.1 经济可行性：

本工具作为一个毕业设计，不需要任何经费支持，所设计实现的解决方案对今后的研究工作有很大的价值。

4.2 技术可行性：

在代码行千万量级的大型代码是非常复杂的。不论是修复故障，或实施一个新的功能，在这样的复杂大型的系统中引起的变化往往有意想不到的影响，因为开发人员不可能时刻保持对代码的完整理解。可视化旨在支持复杂大型代码理解的技术，通过允许开发者查看大型软件代码库的各个层次的抽象，产生一个交互式可视化代码，用户即可以从高层次查代码的层次抽象，也可以进入到每个组件中看到函数。

我们选用 doxygen 解析源码，用强大的可定制图标的可视化工具 echart 可视化代码。从目前本课题的研究工作进展来看，使用这两个工具是相当可行的。

五 研究工作进度安排

理论研究：应包括文献调研，理论推导，数值计算，理论分析，撰写论文等；

实验研究和工程技术研究：应包括文献调研，理论分析，实验设计，仪器设备的研制和调试，实验操作，实验数据的分析处理，撰写论文等。

2016 年 9 月——2016 年 12 月：查找国内外相关论文，阅读有关文献和书籍，了解课题所涉及到的基本概念和方法，明确思路，完成实验环境的熟悉和配置。

2017 年 1 月——2017 年 3 月：深入理解源码编译过程，确定实验方案设计。

2017 年 4 月——2017 年 5 月：实现利用工具，解析源码，得到 XML 文件。再设计实验，处理 XML 文件，得到我们可视化所需的信息。

2017 年 6 月——2017 年 8 月：实现将得到的数据可视化，实现 Linux 源码可视化。

2017 年 9 月——2017 年 12 月：总结成果，撰写论文。

六 预期研究成果

本课题的预期研究成果是通过对 Linux 内核解析，得到源代码的结构信息，并提取代码信息，然后再将其可视化，最后实现一个 Linux 内核代码的可视化工具。

七 本课题创新之处

从研究方法上，本课题从程序员理解代码的所需的认知模型开始着手研究认知模型对于理解程序的重要性，由于大量的研究表明，程序员在理解程序的过程中一般是使用集成元模型，也就是两种模型由上而下和自底向上的结合使用，所以我们的可视化设计结合了这两种模型。

从工具的功能实现上，该工具类比地图功能，提供源代码符号查询、连接查询，源码结构层次查看，源代码查看等功能。

八 研究基础

1.与本项目有关的研究工作积累和已取得的研究工作成绩。

阅读了关于认知模型和代码解析可视化的论文和书籍，积累了相关的基础知识；对理解程序的过程中需要的认知模型有了一定的理解；熟悉了源码解析工具，为解析源码生成所需的数据做好铺垫；对可视化方式做了研究，设计了不同认知模式下的可视化界面。

2.已具备的实验条件，尚缺少的实验条件和解决的途径（包括利用国家重点实验室和部门开放实验室的计划与落实情况）。

已具备的实验条件： Linux 开源内核；解析源码工具。

3. 研究经费预算计划和落实情况。

由于本课题采用的工具均为开源工具，实验室条件足够完备，无需再增加经费开支。

北京理工大學

硕士学位论文开题报告——导师意见

学籍状态	□ 注册 □ 悬置

导师对开题报告的审阅意见：

随着 Linux 内核规模的不断增大，相应的代码维护和程序理解也越来越困难。张晶晶同学提出对 Linux 内核源码进行分析与可视化，帮助程序员从快速理解代码，该选题有很好的应用前景和应用价值。

在开题报告中，张晶晶同学对课题相关研究工作进行了全面和深入的调研，在此基础上对课题所采用的技术路线和实验方法进行了详细的阐述，为后续研究工作的开展奠定了坚实的基础。

开题报告中研究任务明确，研究思路清晰，时间安排合理，具有较高的可行性，同意就此课题进行进一步的深入研究。

导师 签字：

年　　月　　日

说明： 本页全部由指导教师填写。

北京理工大學

硕士学位论文开题报告评审表

学号	2120151086	姓名	张晶晶	导师姓名	刘辉
所在学院	计算机学院		学科、专业	软件工程	
课程学习情况	已修课程学分		30	待修课程学分	0
选题名称	Linux 内核代码分析与可视化技术研究				
课题经费来源	横向经费				
开题报告时间	2016 年 12 月 8 日				

评审组成员		姓　名	职　称	工作单位	签　字
	组长	石峰	教授	北京理工大学	
	组员	刘辉	教授	北京理工大学	
		计卫星	副教授	北京理工大学	
		王一拙	讲师	北京理工大学	
		高玉金	讲师	北京理工大学	

评审组意见：

张晶晶同学的开题报告选题目的明确，实用性强，具有较高的研究价值。报告中阐述了 Linux 内核代码分析与可视化的重要意义，给出了相应的技术路线和设计方案，对其中的关键技术进行了具体的分析。方案合理、可行，在现有的实验条件下可以实现。

该同学在开题过程中表达清晰，回答问题正确，已具备了开题条件，同意进入课题研究工作。

组长签字：　　年　　月　　日