实时、丰富且可组合：超越静态文本编程的特性

Joshua Horowitz [1] **和 Jeffrey Heer** [1]

[1]华盛顿大学，西雅图，华盛顿州

摘要

推动编程超越静态文本代码的努力，一直试图赋予编程多种不同的特性。其中一个长期以来广受认可的特性是活性：在程序编辑过程中，为程序员提供有关程序动态行为的深入反馈。第二个特性长期以来也一直受到探索，但缺乏一个约定俗成的术语，即丰富性：允许程序员通过领域特定表示和交互来编辑程序，而不只是通过文本编辑。在本文中，我们梳理这两个特性之间的关系，并综述体现这些特性的过往工作。我们观察到，将活性与丰富性相结合的系统，往往以牺牲传统编程的一项基本特性可组合性为代价。我们认为，通过结合活性、丰富性和可组合性，编程系统能够更充分地发挥交互式计算的全部潜力，同时不牺牲传统代码的表达能力。关键词：实时编程、可视化编程、终端用户编程、组合、计算笔记本、图形用户界面。

1 引言

自交互式计算诞生以来，人们就一直在探索静态文本代码的替代方案。最早的用户界面包括程序化计算机辅助设计系统 [1] 和流程图编程系统 [2], [3]。随后很快出现了电子表格，至今仍是终端用户编程的典范 [4]。

偏离静态文本有许多原因。最主要的原因是希望让编程变得更加易于使用。编程学习者和潜在的终端用户程序员在接触传统编程语言时会面临严重障碍。也许新的界面能够让更多人释放程序化能力。另一些人则关注静态文本如何迫使所有程序员（甚至是专家）通过抽象、间接的表示形式，与数据可视化和交互式游戏等各种各样的工件进行交互。也许新的界面能够弥合这一鸿沟，让“创作者……与他们所创造的事物建立直接联系” [5]。

受这些动机驱动，研究人员和工具开发者构建了种类繁多的、超越静态文本的编程系统，从电子表格到基于代码块的编辑器，再到新奇的可视化编程语言。显然，静态文本只是广阔的可能编程环境空间中的一个点。为了理解这片由各种点状示例组成、否则可能令人不知所措的云，我们将确定这些设计所推进的基本特性。

我们聚焦于在超越静态文本的探索中逐渐形成的两个核心特性。第一个特性是活性：在程序编辑过程中，为程序员提供关于程序动态行为的深入反馈。这一特性在计算机科学文献中已得到充分确立。[1] 第二个特性尚无名称，尽管许多项目都在隐含地追求它：让程序员能够使用领域特定可视化和交互。我们提出将丰富性 作为一个新的术语，以便更准确地界定这一特性并进行分析。

我们考察这两种特质，调研以不同组合体现它们的系统。在分析过程中，我们遇到了第三种特质，即可组合性：能够自由地将较小的编程产物组合成更大的编程产物，以实现更大的目标。与活性和丰富性不同，这并不是静态文本所缺乏、而交互式编程系统努力为其增添的特质。相反，这是静态文本所具有的一种熟悉特质，而新的编程系统必须努力将其保持下来。

PLATEAU第13届PL与人机交互交叉领域年度研讨会

组织者：Sarah Chasins、Elena Glassman 和 Joshua Sunshine

本作品根据知识共享署名4.0国际许可协议授权。

我们假设，通过结合活性、丰富性和可组合性，编程系统可以在不丧失静态文本特有表达能力的情况下，有意义地扩展其能力。虽然要同时体现所有这些特性颇具挑战，但近期已有多个项目朝着这一目标取得了长足进展。最后，我们将讨论这些系统。

1 活性早在 1990 [6], 年就已被命名；一场以此命名的研讨会自 2013 [7]年以来一直举办至今。

2 活性

编程通常是一门不透明的艺术。当计算机运行程序时，它会沿着控制结构执行，完成各项操作并构造中间值。程序员需要仔细编排这一过程，但其运行行为往往是不可见的。程序员编写程序时，计算机不会提供任何反馈来告知他们程序将如何运行。相反，程序员只能在脑中模拟程序的运行过程，得不到计算辅助。虽然调试器可以提供其中一部分信息，但它们只能通过一个狭窄的观察窗口提供信息，而这个窗口是为特殊的（“调试”）情境设计的。编辑代码时，程序员通常无法看到修改对运行时行为产生的影响。

实时编程旨在摆脱这一现状，创建“即使在编程过程中，也能为程序的动态行为提供即时反馈的编程工具” [8]。“实时”一词在此语境中的使用可追溯至 Tanimoto [6]，，尽管体现不同活性程度的系统早在计算本身诞生之初就已出现。[2]

程序员为什么会想要使用实时编程系统？一项对实时编程相关工作的调查发现，该领域共有 187 篇论文，并归纳出这些论文提出的关于活性的十种动机 [8]。其中包括提高程序员的生产力（通常通过加快编程任务），提升经验较少的程序员的可访问性，以及在阅读现有代码时更容易理解。

2.1 活性的方法

实时编程系统的开发者广泛尝试了实现活性的各种方式。本文根据所采用的一般方法，将若干示例归入几个虽不严谨但颇具帮助的类别。

代码之外的活性。 一些系统能够快速反馈程序的最终输出，而不会重新揭示导致该输出的程序内部信息。这类系统的例子包括应用程序开发中广受欢迎的“实时/热重载”系统 [9] 以及生成艺术中广受欢迎的分屏编辑器（P5.js 网页编辑器 [10]、Victor 中的树木和电子游戏示例 [5] （如图 1 所示）以及 TouchDevelop [11]）。虽然这些系统提供的快速反馈无疑是对缓慢编译运行循环的改进（而且通常需要复杂的技术工作[3]），但这种反馈是可能达到的最粗粒度的活性。我们将在以下类别中描述的活性示例，不仅能让人看到程序的输出，还能了解程序的内部运行。

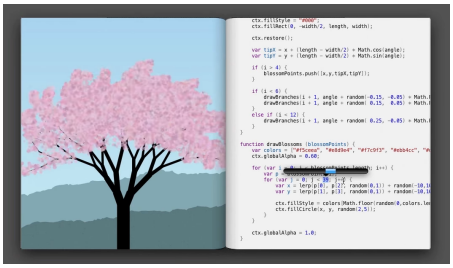


图 1。 Victor 中展示的实时编程界面 [5]。编辑右侧代码时（包括使用图中所示的内嵌滑块等工具），左侧的渲染输出会立即更新。

文本代码中的活性。 一种常见做法是在文本代码编辑器中加入上下文显示，以展示运行时行为。Rauch 等人的《巴比伦式编程》 [13] 概述了截至 2012 年这一类别的技术现状。他们评估了八种现有编辑器 [5]，[12]，[14]–[18]，之后，又将自己的编辑器加入了列表。（Vic- tor [12] 如图 2所示。）2012 年以来，这一类别中值得注意的新成员包括 Xcode 中的 Swift Playgrounds[19] 和 Projection Boxes [20]。

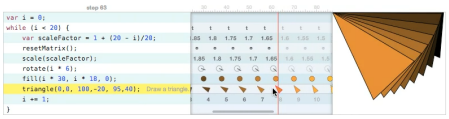


图 2。 Victor 中展示的实时编程界面 [12]。左侧输入文本代码。右侧的时间线展示执行流程，以及中间值和副作用。

2 “实时编程”应与“现场编码”区分开来，后者如今更狭义地指作为音乐或艺术表演一部分的编程。（有关这一差异的更多内容，参见 Rein 等人 [8] 。）3 有关支持活性的技术工作示例，参见 Burckhardt 等人 [11] 。

文本代码单元格之间的活性。 其他编辑器并不将实时反馈嵌入传统代码库，而是提供特殊界面，将代码拆分为“单元格”。这样，这些编辑器便能让用户看到各单元格之间流动的值。这一模式最初的实例是电子表格。近年来，Mathematica [21], Jupyter [22], 和 Observable [23] 等计算笔记本扩展了这一模型，以支持更复杂的计算。更进一步，Natto 将传统笔记本结构重新塑造为二维画布上的节点和连线（如图 3所示） [24].

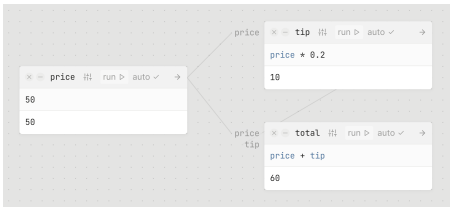


图 3。 Natto 画布，改编自 内置示例。每个窗格运行 JavaScript 代码，这些代码可以引用通过连线连接的其他窗格，并在下方显示其输出值。

结构编辑器中的活性。 结构编辑器 （有时称为 投影式编辑器）不以文本代码或其单元格为基础，而是允许直接与代码的结构化表示进行交互。结构编辑器为实时编程提供了独特的机会。与其将活性改造到已有的文本编辑器中，不如从一开始就将其构建到结构编辑器中。许多项目都利用了这一机会，包括 Subtext [26], Lamdu[25]（如图 4所示）、Enso [27], 和 PANE[28]。

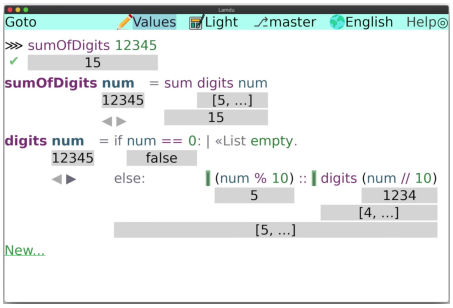


图 4。 Lamdu 的用户界面。代码以模拟传统文本的结构化形式进行编辑。中间值显示在子表达式下方的灰色条中。（引自 Lotem 与 Chuchem [25]。）

动态环境中的活性。 一些早期的实时编程环境，包括 Smalltalk [30], 、Genera 等 Lisp 操作系统 [31], 和 Boxer [29] （如图 5所示），无法明确归入上述类别。在这些环境中，代码与应用程序之间的界限并不清晰。虽然这些环境通常无法实时显示正在运行的代码，但使用它们往往需要执行能够产生即时、可见效果的代码块。

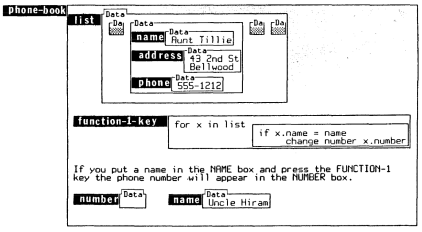


图 5。 使用 Boxer 制作的电话簿。过程、数据结构和界面元素都通过嵌套框表示。（引自 diSessa 与 Abelson [29]。）

3 丰富性

人们在日常生活中使用各种丰富且专业化的计算机界面。我们使用文字处理器撰写文档，使用图形编辑器创建图像，并使用时间线视图进行视频编辑。这些界面围绕各自的领域精心设计，提供支持直接操作和视觉反馈的表示。

将目光转向静态文本编程，这种丰富性便消失了。静态文本无法提供异构的领域特定表示。解析输入、处理图像和构建可视化都是不同的任务，但无论程序员正在处理哪一种任务，其屏幕看起来都一样。领域特定语言可以针对这些不同情境提供独立的方言，但即使使用领域特定语言，编程仍然采取文本形式，失去了日常计算机使用中所体现的直接操作和视觉表示。

学术界和工业界的许多努力，都在探索如何将这类领域特定的丰富性引入 ess

编程。然而，据我们所知，目前还没有一个类似于“活性”的简洁共用术语来描述这种特性。我们提出用“丰富性”来命名这一品质。编程系统只要允许程序员通过针对其所从事的任务、领域和上下文量身定制的可视化与交互来编辑程序，就体现了丰富性。我们对丰富性的检验标准是：在至少某些情况下，系统是否允许程序员用这些定制交互取代编写文本代码。[4] 与活性一样，丰富性旨在成为一个术语：某个东西仅仅在口语意义上让人觉得“丰富”，并不意味着它在我们特定意义上就是丰富的。

为什么程序员会想要一个丰富的编程系统？我们认为，许多支持活性的动机在这里同样适用，包括提升编程系统的生产力、可访问性和理解程度。不过，针对丰富性还可以提出一个更具体的论点，重点在于视觉表示如何带来全新的交流与思维形式。Andersen 等人 [33]写道：“[编]写代码意味着尽可能精确地表达思想……这些思想通常涉及几何关系：表格、对象嵌套、图等。此外，几何结构因问题领域的不同而不同。然而，直到今天，程序员仍然以线性文本表达自己的思想。”类似地，Omar 等人 [34] 写道：“自古以来，图示就在数学思维中发挥着关键作用，事实上，它们早于符号数学的出现。数十年来，面向大众的计算工具和创意工具也一直采用视觉表示和直接操作界面。然而，编程却固执地深陷于文本用户界面之中。”

在深入探讨丰富的编程系统示例之前，我们应先明确“丰富”并不意味着什么：编程系统是“可视化的”，本身并不使其在我们所说的意义上变得丰富。许多可视化编程系统使用视觉结构来表示文本编程中的通用、泛化结构，例如数据流或控制结构。这并不是异质的领域特异性丰富性。[5] 可视化但并不丰富的编程系统包括：基于区块的编程语言，如 Scratch； [35], 节点与连线编辑器，如 Natto； [24] Unreal Engine 的“Blueprints”； [36], 可视化结构编辑器，如 Greenfoot 中的“基于框架”编辑器； [37], 以及经典的“可视化编程语言”，包括 VIPR、 [38], Prograph、 [39], Forms/3 [40], 和 Cube [41]。

在界定“丰富性”的范围时，我们选择了一个尤其狭窄的范畴，以便聚焦于静态文本的替代方案——这些方案能够捕捉我们已逐渐期待应用所具备的丰富、异质性交互性。这个定义仍在不断完善之中。作为我们社区共同进行意义建构的一部分，我们欢迎持续展开讨论。

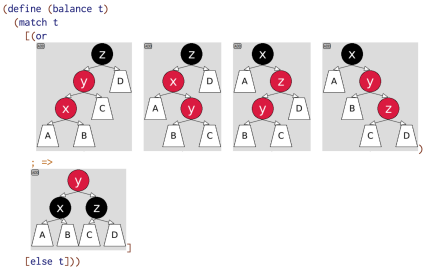
3.1 没有丰富性的活性，没有活性的丰富性

尽管丰富性与活性可能深度交织，但它们仍然是不同的性质。通过考察体现出没有丰富性的活性，或反之的系统，我们可以清楚地说明这一点。

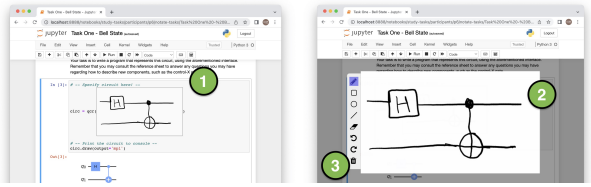
没有丰富性的活性。 我们在第2 节中介绍的所有示例，都经过有意选择而不具备丰富性。其中包括“巴比伦式编程” [13]中所描述的基于文本的编程环境。尽管这些环境能够让人了解代码正在做什么，但代码本身仍然只是作为通用静态文本进行编辑，没有任何领域特定的程序修改方式。

缺乏活性的丰富性。 在丰富但不具备活性的编程系统中，丰富性往往体现为“视觉语法”——可直接编辑的视觉表示，作为传统代码之上的“语法糖”。早期的一个例子是 Erwig 与 Meyer [42], ，其中展示了如何将状态图、AVL 树和基于指针的数据结构图嵌入代码。Andersen 等人 [33]介绍了一个用于 Racket 的可视化宏系统，用户可以通过它定义可交互编辑的视觉语法

4 通过施加这一限制，我们将“只读丰富性”（例如：对实时系统中流经的数据进行领域特定的可视化）排除在丰富性的定义之外。“只读丰富性”的一个很好的例子是 Glamorous Toolkit [32]，这是一个构建于 Pharo Smalltalk 之上的开发环境，提供了可以通过类型特定视图进行扩展的对象检查器。尽管这些检查器在视觉上可以充满活力，甚至具有交互性，但程序无法通过这些交互来构建或修改。因此，Glamorous Toolkit 提供了强大的领域特定的活性，却不具备丰富性。 5 为支持这一划分，Andersen 等人 [33] 写道：“[可视语言] 只提供一组固定的构造，尽管这些构造是可视的——这意味着可视语言未能解决几何思维所具有的特定于问题的性质。”这与 Victor [12], 的观点相呼应，他写道：“传统的可视环境将代码可视化。它们可视化静态结构。但那并不是我们需要理解的内容。我们需要理解代码正在做什么。可视化数据，而不是代码。动态行为，而不是静态结构。”



(a) 一种用于平衡红黑树的过程，使用 Racket 可视化宏系统实现。在这里，嵌入式树图既可作为待匹配的模式，也可作为子表达式。在这两种用途中，它们都可以包含变量。（摘自 Andersen 等人 [33]。）



(b) Arawjo 等人 [45] 的“符号化编程”系统界面。在 (1) 中，画布以内嵌方式显示在 Jupyter 单元格的代码中。选中该画布后会调出 (2) 中的徒手编辑工具。（引自 Arawjo 等人 [45]。）

图 6。 没有活性的两个丰富性示例。

并将其嵌入任意 Racket 代码中。他们还通过树图（见图 6a）、拼图游戏和内嵌表单展示了这一点。Graphite 为 Java 集成开发环境添加了弹出式“调色板”，以便对代码中的字面值（如颜色）进行双向编辑 [43]。MPS（元编程系统）是一款用于开发投影式领域特定语言的商业工作台。MPS 语言通常看起来很像常规代码，但也可以嵌入表格和状态机图等自定义界面 [44]。Arawjo 等人 [45] 探索了将手绘量子电路图整合到 Jupyter 笔记本中的文本代码里，这是作者所称“符号化编程”的一个例子（见图 6b）。尽管这些例子都为静态文本增添了丰富性，但它们仍然是静态的——不会对运行中程序的动态活动做出响应，也不提供实时反馈。

3.2 具有活性的丰富性

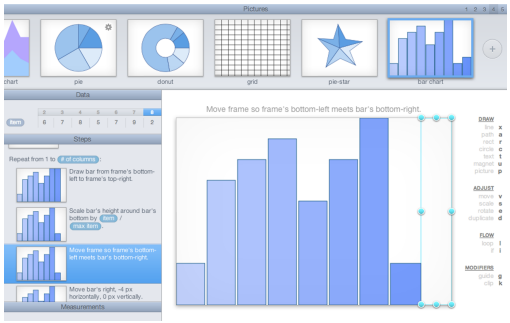
现在我们转向活性与丰富性的交汇点，这种结合带来了协同的可能性。其中最重要的是 示范编程 (PbD) [46]的可能性。在 PbD 系统中，用户通过直接操作与具体对象进行交互。系统能够从针对某个特定输入的“示范”中进行泛化，将该示范转化为可复用的程序。已有项目探索了如何在不同领域实现这一目标，从处理表格数据（Wrangler [47]），到构建定制可视化（绘制动态可视化 [48]，如图 7a所示，以及 Lyra [49]），再到构建完整的 Web 应用程序（Gneiss [50]）。这些项目所采用的机制差异很大，会根据各自领域的需求进行调整。然而，所有 PbD 系统都必须具备活性 和 丰富性。它们必须具备活性，因为示范是在具体输入数据上执行的，用户进行示范时必须显示相应效果。这些效果只有在运行时才能获得，因此纯静态系统无法支持 PbD。PbD 系统还必须具备丰富性，因为直接操作需要领域特定表示和交互。

示范编程是实时且丰富工具中一个尤其令人兴奋的类别，但并非这类工具采用的唯一方法。其他工具仍要求用户以符号方式表达意图，但利用活性和丰富性使这一过程更加容易。此类示例包括 InterState [51]（如图 7b所示）、Apparatus [52], 面向对象绘图 [53], 以及各种正则表达式开发工具 [54]–[56]。这些工具使用领域特定可视语法，例如 §3.1“无活性的丰富性”标题下的那些语法，但它们将实时反馈整合进这些语法中。

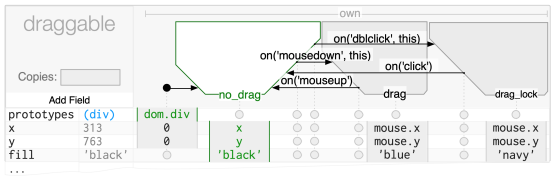
4 可组合性

我们先前的讨论中隐含着一种模式。在讨论“没有丰富性的活性”和“没有活性的丰富性”（见 §3.1）时，大多数示例系统都将自身定位为通用系统，大致相当于传统的通用编程语言，例如 Python。

在我们最近关于“具有活性的丰富性”的讨论中却并非如此（见第3.2节）。该节介绍的工具面向特定领域。事实上，这些工具的优势很大程度上正源于这种特定性。每一种工具都以量身定制的直接交互，取代了通用的符号编程流程，



(a) 绘制动态可视化中的界面。构建了一个条形图结合了在画布视图中的直接操作，以及对左侧脚本的显式编辑面板。（摘自 Victor [48]。）



(b) InterState 对象的编辑器。状态和转换会显示在上方的面板中，并可在那里进行编辑。值不同状态下属性的值在下方编辑，具体的实时值则显示在左下角。（引自 Oney 等人 [51]。）

图 7。两个示例　　体现活性的丰富性：第一个通过编程实现　　-通过演示，后者不是。

专门服务于其所在领域。

然而，这留下了一个重要问题：如何将这些强大但专门化的工具结合起来，以解决现实世界编程中的目标？现实世界编程通常需要协同执行多项任务。例如，假设一名数据分析师想要了解产品发布进展得如何。他们可能需要：1. 从 API 获取数据；2. 将这些数据从半结构化 JSON 重新格式化为表格格式；3. 进一步处理表格数据；4. 构建图表以展示这些数据中的模式。每个步骤都可以由针对性强、实时且丰富的工具提供支持。例如，Gneiss [50] 可以支持步骤 1 和 2，Wrangler [47] 可以支持步骤 3，而 Lyra [49] 可以支持步骤 4。分析师可以在什么样的环境中流畅地将这些工具结合起来使用？

我们所追求的是可组合性：将较小的编程产物自由组合成更大的编程产物，以实现更宏大的目标。可组合性是传统编程中一种普遍存在的特质：无论是编写会调用多个库的 Python 脚本，还是仅仅将 `if` 和 `for` 嵌套在彼此之中，都能体现这一点。归根结底，自由组合构建模块的能力，正是编程拥有无限表达潜力的原因。

我们在第3.1 节考察的具有活性但不丰富、以及丰富但不具活性的系统，遵循传统编程的模式，并保持了可组合性。然而，值得注意的是，第3.2 节讨论的具有活性且丰富的系统，都是作为独立应用实现的，而不是作为某个更大型编程系统中的可组合组件。这一趋势表明，当活性与丰富性结合时，实现可组合性会面临困难。操作系统级特性和编程平台（直到最近）都无法实现丰富接口的实时组合。遗憾的是，这使得具有活性且丰富的工具在需要组合多个工具的现实场景中更难使用，例如上述数据分析师所面临的场景。

本文提出的主要问题是：在不放弃可组合性这一关键特性的前提下，编程如何获得活性和丰富性这两项理想特性？在本节中，我们首先讨论几种当前的方法；我们认为，这些方法无法提供完整的可组合性。随后，我们将介绍两个近期的系统，它们从一开始就成功结合了这三种特性。

4.1 当前方法：手动集成

我们在第§3.2 节讨论的实时、丰富且面向领域的工具，通常会以某种形式与外部世界集成，即便只是为了演示。例如，可以复制和粘贴文本片段：将正则表达式复制到正则表达式开发工具中，或将生成代码从 Wrangler 中复制出来（再复制回去） [47]。另一种形式是导入和导出文件，例如将表格数据加载到 Wrangler 中，或从 Wrangler 中导出表格数据。无论哪种情况，我们都将其称为手动集成，因为这要求用户手动在工具及其周围环境之间传递变更。从某种程度上说，手动集成确实可行。用户可以在工具与代码库之间来回复制程序片段和数据，从而完成某种工作流。然而，这种

工作流仍有很大的改进空间。

最重要的是，缺乏整体活性。如果某种情境需要使用多个组合工具，那么在一个工具中的操作可能产生下游影响，进而影响另一个工具。例如，如果上述数据分析师改为从新的 API 获取数据，这可能会影响他们整理数据的方式。要让整个工作流在整体意义上保持活性，所做更改的影响就应尽可能迅速地在下游工具中显现，同时尽可能减少令人分心的操作。然而，在手动集成系统中，程序员必须手动将对某个工具输入和输出的更改传递下去。虽然这些工具内部可能具有活性，但工具之间的边界并不具备活性。

手动集成还会带来其他代价。它缺乏并置。使用单独的工具意味着必须离开程序员惯用的环境，进入一个新的空间。这可能使人迷失方向、分心。此外，它还缺乏完整的持久性。例如，程序员使用正则表达式辅助工具时，或许会整理一组测试输入，但当正则表达式被复制粘贴回代码中时，这组内容并不会被保留下来。

这些问题在手动集成工具时会累积成高昂的成本。由于这种成本是按工具单独产生的，它会抑制小型工具的开发，尽管撇开集成成本不谈，由多个小型工具组成的组合将带来更大的灵活性和组合可能性。在集成成本高昂的环境中，只有大型单体工具才能生存下来。

4.2 当前方法：工具包

如果手动集成会带来高昂的成本，那么一种自然的替代方案就是将用户在完成某项更大规模任务时可能需要的多个具备活性且内容丰富的工具，整合到一个工具中。

这种方法的一个很好的例子是 Gneiss[50]，，如图 8 所示。Gneiss 是一个用于制作交互式数据驱动 Web 应用程序的工具。它通过三个窗格将三个独立工具组合在一起：API 查询工具、电子表格和界面构建器。这些工具支持流畅的工作流：用户构建 API 调用，将响应中的各个部分拖入电子表格，并基于该电子表格构建界面。数据会在这些工具之间自动流动，从而实现一种集成式的活性体验；如果这些工具是手动集成到静态代码库中，这种体验将无法实现。Gneiss 有力地展示了集成式活性的优势——活性不仅存在于工具内部，也存在于工具之间。

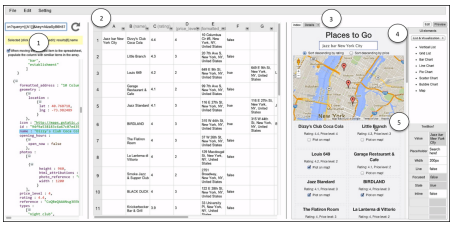


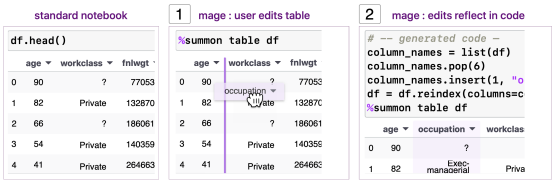
图8。 Gneiss 界面由三个窗格组成：(1) 用于构建 API 调用并显示结果。这些结果可以拖入 (2)，即电子表格界面。(3) 是一个由电子表格中的数据驱动的直接操作式 Web 界面构建器。（引自 Chang 和 Myers [50]。）

然而，Gneiss 仅将这种集成式活性提供给一套特定且固定捆绑的工具。这意味着，如果用户的工作流与 Gneiss 创作者设想的工作流相近却并不完全相同，Gneiss 将无法满足他们的需求。如果用户想基于导入的 CSV 数据构建界面，而不是使用来自 API 的 JSON，该怎么办？如果用户想根据自己的数据构建定制图表，而不是构建 Web 应用，又该怎么办？即使存在支持这些步骤的工具，Gneiss 用户也无法将其纳入其中。

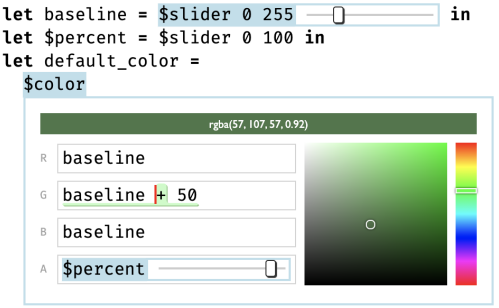
反方向上也存在类似的限制。为了构建 Gneiss，其创作者投入了大量工作来设计和实现三个精巧的实时工具。这些工具的领域想必比 Gneiss 所围绕设计的特定工作流更为广泛。然而，由于这些工具被内置于 Gneiss 中，它们无法被带入新的环境或上下文。

如果按照现代应用软件的标准来评判 Gneiss，这些限制并不令人意外。在图形软件界面的世界中，我们并不期望看到开放式可重新配置性和模块化。[6] 然而，从编程系统的角度来看，这些限制却十分显著。如果用传统代码来表示 Gneiss 的功能，它们看起来会像一个固定脚本，调用三个以特定方式连接的库。用户对 Gneiss 的

6 A 对此的一个部分例外是支持插件或扩展机制的应用程序，例如网络浏览器 a nd pla类似 tform 的创意工具。



(a) mage 工作流：从一个笔记本单元格中召唤出实时工具。实时工具中的直接操作编辑会以代码形式写回它们所召唤自的单元格。（摘自 Kery 等人 [59]。）



(b) Hazel 中 Livelits 的一个使用示例。颜色选择器和各种滑块都是 Livelits，它们嵌入 Hazel 代码中（也相互嵌入）。（摘自 Omar 等人 [34]。）

图 9。 结合活性、丰富性和可组合性的两个系统。

交互会修改这些库调用中的空白槽位。然而，用户无法导入其他库，也无法将现有库重新排列成新的结构。传统编程中存在的开放式可组合性，在图形界面的世界中并不存在，即使是像 Gneiss 这样实时且丰富的界面也不例外。

4.3 当前方法：非持久化工具

乍看之下，Observable [23]似乎将活性和丰富性与可组合性融为一体。作为一种基于单元格的计算笔记本，它无疑体现了活性。此外，Observable 单元格可以将任意 HTML 渲染为其输出的一部分，其中包括交互式小部件，似乎体现了丰富性。由于这些小部件中的交互可以影响单元格的输出，值可以从一个交互式小部件的输出流向另一个交互式小部件的输入。这似乎使活性、丰富的工具能够实现组合。

尽管 Observable 确实接近这一目标，但它未能实现这一目标。阻碍其实现这一目标的最重要限制在于，尽管 Observable 单元格可以渲染用户界面，但该用户界面无法持久地修改底层程序。也就是说，即使 Observable 单元格可以将任意丰富的工具作为其输出的一部分进行渲染，与该工具的交互也无法“保存”回笔记本，并且其效果在笔记本重新加载时总会消失。如果编程工具要成为*编写*程序过程的一部分，而不只是被短暂使用，那么丰富的用户界面就必须能够修改底层程序。

活跃、丰富且具备持久性的工具，其吸引力之强，足以促使包括 Observable 在内的多个平台将这类工具作为一次性功能加入其中。Observable 提供了“数据表单元格”功能，允许用户在普通单元格的位置嵌入一个活跃、丰富且可直接操作的表格编辑器 [57]。类似地，计算笔记本 Hex 提供了多个“无代码”单元格：用于构建图表、进行数据透视和数据筛选，以及执行其他特定任务的活跃、丰富工具 [58]。在这两种情况下，这些工具都是专门集成到各自平台中的，并且可以访问第三方工具无法获得的持久性功能。

4.4 活性、丰富性与可组合性

我们已经看到，手动将工具组合在一起的尝试无法保持跨工具边界的活性，而将工具组合成预制套件的尝试又无法为用户提供开放式可组合性。要克服这些局限，系统需要将持久、丰富的接口以开放式组合的方式整合进一个集成式活系统中。最近有两个项目实现了活性、丰富性与可组合性的三重交汇。

mage。 mage 是“一个用于在笔记本中实现图形用户界面与代码工作之间平滑转换的 API”，以 Jupyter Notebook 扩展的形式实现 [59]。使用 mage 构建的工具可以从 Jupyter Notebooks 的单元格中启动。工具会与其单元格一同呈现，并且可以读取和写入该单元格中的代码。这一工作流如图 9a所示。mage 通过多种工具进行了演示，包括图表构建器、图像编辑器，以及可将直接操作

操作转换为数据框操作代码的表格编辑器。

mage 是实时的：与工具交互会导致其所在单元格的代码被重写并自动重新运行。包含工具的单元格可以快速重新运行，以反映其输入的任何变化。mage 是丰富的：使用 mage API 实现的工具可以利用 Web 平台提供的所有功能，来提供领域特定可视化和交互。工具可以将代码写回其 Jupyter 单元格宿主，使其成为持久的编程工具，而不仅仅是短暂的运行时界面。mage 是可组合的：这种可组合性源于 mage 嵌入 Jupyter Notebook。通过将多个 mage 工具分别放置在不同单元格中，并使用变量在它们之间传递数据，可以协同使用这些工具。

Livelits。 Livelits（实时字面量）是 Hazel 的扩展， [60], Hazel 是“一种围绕空洞驱动开发而设计的实时函数式编程环境”， [34]。Hazel 本身是一种实时结构化编辑器，用于模拟传统的静态类型函数式代码。Livelits 在此基础上加入了“持久嵌入代码中的用户定义图形用户界面” [34] 。Livelits 通过一些这样的图形用户界面进行演示，包括滑块、颜色选择器、表格字面量和图像编辑器。其中一些图形用户界面在图 9b中展示了协同工作的效果。mage 工具存在于 Jupyter 单元格这一层级，只允许“扁平”的嵌入结构；而 livelits 则存在于 Hazel 抽象语法树节点这一层级。这意味着 livelits 与文本形式的 Hazel 代码可以彼此递归地、任意嵌套。

Livelits 是活的：这种活性最终源自 Hazel，就像 mage 的活性源自 Jupyter 一样。Hazel 的活性在几个方面比 Jupyter 更深入。在 Hazel 中，任意子表达式的值都可以获得实时反馈，而不仅限于单元格层面。此外，Hazel 以响应式方式运行，会自动传播变化，无需用户明确地重新运行单元格。Livelits 是丰富的：与 mage 工具一样，它们可以访问 Web 平台，从而支持多样且生动的界面。它们会将自身状态（任意可序列化模型）直接持久化到程序的抽象语法树中，而不是像 mage 工具那样生成并重新解析文本代码。Livelits 是可组合的：它们可以出现在 Hazel 表达式能够出现的任何位置。Livelits 不仅可以通过变量赋值链接起来（类似于在 Jupyter 单元格中组合 mage 工具），还可以直接嵌入彼此之中、嵌入函数定义等结构中。

讨论。 mage 和 livelits 实现了我们认为此前尚不可能实现的功能：将领域特定的交互式编程工具集成到通用编程环境中，并提供完整的活性，以支持示范编程等技术。想象一下，我们在第 §3.2 节中考察过的那些实时且丰富的工具——从可视化编辑器到直接操作式数据整理工具——如今被重新构建为 mage 或 livelits 中的工具。以这种方式构建后，它们便可以融入更强大的工作流，与其他工具或传统文本代码结合使用。mage、livelits，或类似的平台可以通过实现组合，为实时且丰富的工具释放新的价值。

mage 和 Livelits 存在相互对立的局限性，阻碍了这一愿景的实现。mage 工具灵活而强大，但由于它们是在 Jupyter 单元格层面调用的，因此无法嵌入循环或函数定义等通用编程构造中。此外，虽然 Jupyter 在数据工作等上下文中非常实用，但目前尚不清楚如何将 mage 基于 Jupyter 的方法推广到应用程序开发等其他上下文。另一方面，Livelits 作为一等语言构造存在，可以嵌入 Hazel 语言的任何位置。然而，Hazel 的严格类型约束限制了它们的灵活性。我们讨论过的许多工具，例如 Gneiss 的子工具和 Wrangler，都无法作为 Livelits 存在，因为它们的行为无法用 Hazel 的类型系统表达。正如“实时字面量”这一名称所暗示的那样，迄今为止，Livelits 一直局限于滑块和颜色选择器等简单小部件，用于构造简单值。

目前，mage 和 Livelits 都尚未向公众开放。我们想知道，一个公开可用的组合平台是否能够促进对实验性实时与丰富工具的研究。假设一位研究人员有一个新颖的实时与丰富工具的想法。无论该工具在其任务中多么有效，要评估它，都需要构建足够的配套基础设施，以便在端到端工作流中展示该工具。这可能涉及大量工作，而这些工作与他们正在探索的新颖交互无关。如果研究人员能够在一个类似于 mage 或 Livelits 的系统中实现其工具，他们就可以在配套工具和常规代码逃生舱的环境中对其进行测试。这样一来，一个公开可用的组合平台就可以

降低实时与富工具研究的入门门槛，并加速该领域的创新。

5 结论

活性与丰富性为超越静态文本编程的探索提供了互补的方向。活性让程序员能够了解程序的内部行为，并对其编辑做出反应；丰富性则让程序员能够通过与其工作领域相匹配的视觉表示来编辑程序。这些方向可以分别推进，但将二者结合起来会带来更多可能性，从而实现示范编程等新技术。

尽管实时且丰富的编程界面具有巨大潜力，但当它们以独立应用程序的形式实现时，其实际效用会受到限制。手动集成和捆绑等主流方法无法复现传统代码的自由形式的可组合性，至少无法在不牺牲整体活性的情况下做到这一点。

mage 和 Livelits 旨在解决这种可组合性缺失的问题。它们遵循一种共同模式，分别为现有的通用编程环境（Jupyter 和 Hazel）扩展实时且丰富的工具。其他实时编程环境——从“巴比伦式编程”中讨论的基于文本的环境 [13] 到 Observable 等笔记本 [23] 以及 Natto 等视觉编辑器 [24], 是否也能以类似方式扩展？或者，提出一个更具雄心的目标：能否在这些环境之间共享工具，从而让人们能够在探索多样化工具的同时，也探索多样化的环境？

mage 和 Livelits 才刚刚开始探索活性、丰富性与可组合性三者交汇处的设计空间。我们最希望的是，通过阐明这组特质，鼓励更多人投身这一新兴领域。

致谢

我们感谢 Geoffrey Litt、Paula Te、华盛顿大学交互数据实验室以及 PLATEAU 研讨会社区提供的宝贵反馈。本研究得到了摩尔基金会软件资助的支持。

参考资料

[1] I. E. Sutherland，《Sketchpad》，载于*1963*年*5*月*21*日至*23*日春季联合计算机会议论文集——*AFIPS ’63*（春季），ACM Press，1963年。[2] T. O. Ellis、J. F. Heafner 和 W. L. Sibley，《GRAIL 项目：人机交流实验》，1969年。[3] T. Ellis、J. Heafner 和 W. Sibley，*GRAIL* 语言与操作。兰德公司，1969年。[4] B. A. Nardi，编程小事（A Small Matter of Programming）。英国伦敦：MIT Press，1993年7月。[5] B. Victor，《基于原则的发明》，https://vimeo.com/36579366，发表于加拿大大学软件工程会议（CUSEC），2012年。[6] S. L. Tanimoto，《Viva：一种用于图像处理的可视语言》，可视语言与计算期刊，第1卷，第2期，第127–139页，1990年6月。[7] *2013*年实时编程研讨会。[8] P. Rein、S. Ramson、J. Lincke、R. Hirschfeld 和 T. Pape，《探索式与实时编程和编码》，编程的艺术、科学与工程，第3卷，第1期，2018年7月。[9] D. Abramov，*Live React*：结合时间旅行的热重载，2015年7月。[10] p5.js，*P5.js* 网页编辑器。[11] S. Burckhardt、M. Fahndrich、P. de Halleux、等，《它活了！用户界面编程中的持续反馈》，*ACMSIGPLAN Notices*，第48卷，第6期，第95–104页，2013年6月。[12] B. Victor，可学习编程，http://worrydream.com/LearnableProgramming/，2012年。[13] D. Rauch、P. Rein、S. Ramson、J. Lincke 和 R. Hirschfeld，《巴比伦式编程：将实时示例集成到通用源代码中的设计与实现》，编程的艺术、科学与工程，第3卷，第3期，2019年2月。

[14] J. Edwards, 《以示例为中心的编程》, *ACM SIGPLAN Notices*, 第39卷, 第12期, 第84–91页, 2004年12月。[15] C. Granger, *LightTable*, http://lighttable.com/, 2022年。[16] T. Imai、H. Masuhara 和 T. Aotani, 《Shiranui：支持单元测试的实时编程》, 载于*2015*年 *ACM SIGPLAN* 系统、编程、语言与应用国际会议软件与人文分会论文集, ACM, 2015年10月。[17] S. Kasibatla 和 A. Warth, 《Seymour：面向课堂的实时编程》, https://harc.github.io/seymou

r-

live2017/, 《实时编程研讨会》（LIVE）2017, 2017。[18] T. van der Storm 和 F. Hermans, 《实时字面量》, https://homepages.cwi.nl/~storm/livelit/livelit.html, 《实时编程研讨会》（LIVE）2016, 2016。[19] Apple Inc., 《Wwdc 2014》, https://www.youtube.com/watch?v=w87fOAG8fjk, 2014。[20] S. Lerner, 《Projection Boxes：面向实时编程的即时可重配置可视化》, 载于《2020年CHI计算机系统中的人为因素会议论文集》, ACM, 2020年4月。[21] S. Wolfram, 《Mathematica之书》, 第5版。伊利诺伊州尚佩恩：Wolfram Media Inc, 2003年8月。[22] T. Kluyver、B. Ragan-Kelley、F. Pérez等, 《Jupyter Notebooks——用于可复现计算工作流的出版格式》, 载于《电子出版国际会议》, 2016年。[23] Observable Inc., 《Observable——探索、分析和解释数据。以团队形式。》https://observablehq.com/, 2022年。[24] P. Shen, 《欢迎！– natto》, https://natto.dev/, 2022年。[25] E. Lotem 和 Y. Chuchem, Lamdu, https://www.lamdu.org/, 2022年。[26] J. Edwards, 《Subtext》, 载于《第20届ACM SIGPLAN面向对象编程、系统、语言与应用会议论文集》, ACM, 2005年10月。[27] Enso, Enso, https://enso.org/, 2022年。[28] J. Horowitz, 《Pane：使用可见数据进行编程》, http://joshuahhh.com/projects/pane/, 《实时编程研讨会》（LIVE）2018, 2018年。[29] A. A. diSessa 和 H. Abelson, 《Boxer：一种可重构的计算媒介》, 《ACM通讯》, 第29卷, 第9期, 第859–868页, 1986年9月。[30] A. Goldberg 和 D. Robson, 《Smalltalk-80：其语言及实现》。Addison-Wesley Longman Publishing Co., Inc., 1983年。[31] J. Walker、D. Moon、D. Weinreb 和 M. McMahon, 《Symbolics Genera编程环境》, 《IEEE软件》, 第4卷, 第6期, 第36–45页, 1987年11月。[32] T. Girba, Glamorous Toolkit。[33] L. Andersen、M. Ballantyne 和 M. Felleisen, 《为文本代码添加交互式可视语法》, 《ACM编程语言学报》, 第4卷, 第1–28页, 2020年11月。[34] C. Omar、D. Moon、A. Blinn、I. Voysey、N. Collins 和 R. Chugh, 《使用实时图形用户界面填充类型化空洞》, 载于《第42届ACM SIGPLAN编程语言设计与实现国际会议论文集》, ACM, 2021年6月。[35] M. Resnick、J. Maloney、A. Monroy-Hernández等, 《Scratch》, 《ACM通讯》, 第52卷, 第11期, 第60–67页, 2009年11月。[36] I. Epic Games, Blueprints可视化脚本。[37] M. Kölling、N. C. C. Brown 和 A. Altadmri, 《基于帧的编辑》, 载于《初级和中级计算教育研讨会论文集》, ACM, 2015年11月。[38] W. Citrin、M. Doherty 和 B. G. Zorn, 《一种完全可视的面向对象编程语言的设计》, 1994年。[39] P. Cox、F. Giles 和 T. Pietrzykowski, 《Prograph：将编程从文本束缚中解放出来的一步》, 载于 *[《论文集》] 1989* IEEE视觉语言研讨会, IEEE Comput. Soc. Press。[40] M. M. Burnett 和 A. L. Ambler, 《声明式视觉编程语言中的交互式可视数据抽象》, 《视觉语言与计算》, 第5卷, 第1期, 第29–60页, 1994年3月。

[41] M. Najork 和 S. Kaplan，《CUBE语言》，载于*1991*年*IEEE*视觉语言研讨会论文集，IEEE Co
mput. Soc. Press。[42] M. Erwig 和 B. Meyer，《异构视觉语言——整合视觉编程与文本编程》，载于视
觉语言研讨会论文集，IEEE Comput. Soc. Press。[43] C. Omar、Y. S. Yoon、T. D. LaToza 和 B. A.
Myers，《主动代码补全》，载于*2012*年第*34*届国际软件工程会议（*ICSE*），IEEE，2012年6月。[44]
JetBrains s.r.o.，*MPS*：*JetBrains*的领域特定语言创建器，https://www.jetbrains.com/m

ps/，

2022年。[45] I. Arawjo、A. DeArmas、M. Roberts、S. Basu 和 T. Parikh，《笔记本环境中的符号
化编程：量子电路案例研究》，载于第*35*届年度*ACM*用户界面软件与技术研讨会，ACM，2022年10月。
[46] A. Cypher 编，《看我做什么：通过示范编程》，麻省理工学院出版社，1994年。[47] S. Kandel、
A. Paepcke、J. Hellerstein 和 J. Heer，《Wrangler》，载于*SIGCHI*计算机系统人为因素会议论文集，
ACM，2011年5月。[48] B. Victor，《绘制动态可视化》，https://vimeo.com/66085662，于2013年2月
1日在斯坦福大学人机交互研讨会上演讲，2013年。[49] A. Satyanarayan 和 J. Heer，《Lyra：交互式可
视化设计环境》，*ComputerGraphics Forum*，第33卷，第3期，第351–360页，2014年6月。[50] K. S.-P.
Chang 和 B. A. Myers，《使用电子表格创建交互式Web数据应用》，载于第*27*届年度*ACM*用户界面软件
与技术研讨会论文集，ACM，2014年10月。[51] S. Oney、B. Myers 和 J. Brandt，《Interstate》，载于
第*27*届年度*ACM*用户界面软件与技术研讨会论文集，ACM，2014年10月。[52] T. Schachman 和 J.
Horowitz，*Apparatus*：用于创建交互式图示的混合图形编辑器与编程环境。[53] H. Xia、B. Araujo、
T. Grossman 和 D. Wigdor，《面向对象绘图》，载于*2016*年*CHI*计算机系统中的人为因素会议论文集，
ACM，2016年5月。[54] F. Dib，*Regex101*：构建、测试和调试正则表达式。[55] S. Toarca 和 P.
Prelich，*Debuggex*：在线可视化正则表达式测试器。*JavaScript*、*Python* 和 *PCRE*。[56] gskinner，
Regexr：学习、构建、测试正则表达式。[57] Observable Inc.，使用数据表单元格 / *Observable* /
Observable 快速探索和分析数据，
https://observablehq.com/@observablehq/introducing-data-table-cell，2022年。[58] I. Hex
Technologies，*Hex*——与团队协作，更高效地处理数据。[59] M. B. Kery、D. Ren、F. Hohman、D.
Moritz、K. Wongsuphasawat 和 K. Patel，《计算笔记本中的代码与图形工作之间的流畅转换》，载于
第*33*届年度*ACM*用户界面软件与技术研讨会论文集，ACM，2020年10月。[60] C. Omar、I. Voysey、R.
Chugh 和 M. A. Hammer，《带类型化空洞的实时函数式编程》，*ACM*编程语言论文集，第3卷，第1–
32页，2019年1月。