Software Engineering and Applications 软件工程与应用, 2024, 13(5), 651-659
Published Online October 2024 in Hans. https://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2024.135067

Hans汉斯

程序设计方法的范式演进及发展趋势

徐明毅

武汉大学水利水电学院，湖北 武汉

收稿日期：2024年9月1日；录用日期：2024年10月14日；发布日期：2024年10月23日

摘 要

文章回顾了软件开发方法的范式演进，将之大致划分为三个大的阶段。首先从基于机器指令编程发展到使用计算机高级语言编程，这隔离了指令集的差异，使得面向过程编程得到广泛应用。然后随着软件规模的膨胀，为提高数据的封装性和代码的简洁性，发展了基于对象和面向对象编程，而对不同对象的共同属性进行操作，就称之为面向概念编程，这可协调封装性和灵活性的矛盾。为消除编程语言之间的差异，程序设计中大量采用二进制层面的面向对象编程，即面向组件编程，如果软件的组件可在运行时灵活替换，软件运行环境也从单机状态扩展到整个互联网，则称之为组件市场编程。进入人工智能时代，大型语言模型也将迅速应用于软件开发领域，编程助手可将人类自然语言翻译为程序代码，编程将从面向程序语言转为基于自然语言，隔离了编程语言的差异，使每个人都可成为程序员。编程助手智能化程度提高后就成为数字员工，将进入计算机自主进化编程阶段。程序员还可训练出“数字替身”，让软件开发更为轻松惬意。

关键词

程序设计方法，面向过程编程，面向对象编程，大型语言模型，编程助手，智能体

The Paradigm Evolution and Development Trend of Programming Methods

Mingyi Xu

School of Water Resources and Hydropower Engineering, Wuhan University, Wuhan Hubei

Received: Sep. 1st, 2024; accepted: Oct. 14th, 2024; published: Oct. 23rd, 2024

Abstract

This paper reviews the paradigm evolution of software development methods, which can be roughly divided into three major stages. Firstly, the development from machine instruction based programming to programming using advanced computer languages has isolated the differences in instruction

文章引用: 徐明毅. 程序设计方法的范式演进及发展趋势[J]. 软件工程与应用, 2024, 13(5): 651-659.
DOI: 10.12677/sea.2024.135067

sets, making procedural programming widely used. Secondly, with the expansion of software scale, in order to improve data encapsulation and code simplicity, object-based and object-oriented programming have been developed. Operating on the common properties of different objects is called concept-oriented programming, which can reconcile the contradiction between encapsulation and flexibility. Lastly, in order to eliminate the differences between programming languages, binary level object-oriented programming is widely used in program design, which can be called component oriented programming. If the software components can be flexibly replaced at runtime, and the software running environment extends from the stand-alone state to the entire Internet, it can be called component market programming. Entering the era of artificial intelligence, large language models will also be rapidly applied in the field of software development. Programming copilots can translate human natural languages into program codes, and programming will change from programming languages oriented to natural languages based, isolating the differences between programming languages and enabling anyone to become a programmer. After the intelligence level of programming copilots improves, they will become digital employees and enter the stage of computer autonomous evolution programming. Programmers can also train "digital avatars" to make software development easier and more enjoyable.

Keywords

Programming Methods, Procedure-Oriented Programming, Object-Oriented Programming, Large Language Model, Programming Copilot, Agent

1. 引言

信息技术在现代社会中须臾无离，其发展包括硬件和软件两方面，硬件是躯壳，软件是灵魂，先进的计算机系统必须要有与之配套的软件系统才能发挥作用。软件是在数字世界中对客观世界中问题空间与解空间的一种映射，客观世界是不断变化的，软件也体现出构造性和演化性的特征[1] [2]。开发软件，即进行程序设计，随着计算机硬件技术的发展，程序设计方法也在不断发展，其历史演进有迹可循，未来发展趋势则更引人注目[3] [4]。

早期的计算机程序采用机器语言和汇编语言，机器语言直接用二进制指令编码，难于记忆，只适用于简单的程序设计，而汇编语言则用有意义的标识符来代替机器指令，使得编写较长的程序成为可能。后来设计出了计算机高级语言，使用变量、标识符、表达式等概念作为语言的基本构造，因此可以在一个更高的抽象层次上进行程序设计，随后结构化过程式程序设计(Structured Procedural Programming, SPP)应运而生，在解决中等复杂程度的问题时表现出卓越的性能[5] [6]。随着程序规模与复杂度的提升及软件行业的产业化，要求进一步加强软件的重用性，需要同时重视程序中的数据结构和对数据的操作，于是面向对象的程序设计(Object Oriented Programming, OOP)方法逐渐显示出生命力，成为日渐流行的软件开发方法[7] [8]。将具有类似操作属性的对象进一步归类，就发展出面向概念的程序设计(Concept Oriented Programming, COP)，或也可称为面向方面编程(Aspect Oriented Programming, AOP)，这使得程序设计的抽象层次进一步提高[9] [10]。为统一跨语言的复用操作，在二进制层面上制定标准，就形成了组件式编程[11]-[13]。如果类似的组件可在运行时灵活替换和组装，就是组件市场编程，而在互联网开放、动态和多变环境下的软件系统，则可称为网构软件[14] [15]。

在进入人工智能时代的当下，随着以 ChatGPT 为代表的大型语言模型(Large Language Model, LLM)的智能提升，程序设计方法也将面临新的重大突破，即将进入自然语言编程和自动化编程的新阶段。

2. 机器指令编程

机器指令是计算机硬件和软件的沟通桥梁，也是软件的起点，在此基础上不断演化出越来越庞大、复杂、抽象的软件系统。当今计算机的指令架构分为复杂指令集 CISC 和精简指令集 RISC 两大种类，分别以 x86 和 ARM 为代表，一个主要用于 PC 机和服务器领域，一个主要用于手机领域，但两者也试图向对方领域进行渗透。另外，还有属于精简指令集的 RISC-V 架构正在蓬勃兴起，我国具有自主性和兼容性的龙架构(Loong Arch)指令集也在推广应用。

在计算机高级语言出现之前，计算机编程只能基于机器指令(或者说是机器语言)，如果将指令简单加以包装和变形，定义为助记符形式，就是汇编语言编程。这时，表达软件模型的基本概念是机器指令，表达模型处理逻辑的主要机制是顺序和转移。显然，这一抽象层次是比较低的，难以编写复杂的程序，编写和调试都较为困难。机器指令编程的优点是底层、高效，在极为要求效率的情况下可采用该方法编程，但在软件功能不断扩充的情况下就显得极为吃力，直接基于机器指令开发应用系统几无可能。因此，在 20 世纪 60 年代到 70 年代，计算机高级语言应运而生，如 FORTRAN 语言、PASCAL 语言、C 语言等，通过高级语言对计算机的底层指令进行抽象和封装，使编程的难度下降，扩大了计算机的应用范围，同时程序的规模也逐渐膨胀。1968 年，“软件工程”这一概念被首次提出，使得程序设计正式成为现代工业的一个重要分支[1]。随后出现了结构化程序设计技术，实现了模块化的数据抽象和过程抽象，使得软件工业迅速兴起并发展起来。

3. 面向过程编程

一般认为，程序 = 数据结构 + 算法[5]，面向过程编程就是在计算机高级语言已经定义的基本数据类型下，侧重于算法实现从而完成软件开发。基本思想是详细分析实际映射问题的求解方法和实现步骤，将程序分解为四种基本结构形式：顺序结构、选择结构、循环结构和子程序(过程/函数)，遵循一定的便于多人协作的软件工程方法，编写出计算机高级语言程序代码并测试运行，也就是所谓的结构化编程。

过程化程序设计的原则是从上向下、精益求精和模块化，即“瀑布”式开发方法[6]。开发步骤一般是：首先要明确问题，确定用户需求，然后根据功能、过程划分子系统，并完成功能模块的细分；采用单入口/单出口的控制结构，进一步对每个功能模块进行算法设计和优化；然后进入编码阶段，完成各个功能模块，再进行调试，包括程序调试、功能调试、系统调试；最后将软件系统投入运行，并进行后期的软件维护工作。

在软件工程不断发展的过程中，面向过程的程序设计方法所包含的问题也逐渐暴露，其中软件的生产效率低下、维护困难等最为常见。因为面向过程的程序设计的抽象性较低，便于实现的优点，所以比较适用于处理一些比较简单、规模较小的问题。但在数据复杂、功能需求不断调整的情况下，面向过程的程序设计在代码膨胀的过程中，由于粒度较细，数据交织严重，往往“牵一发而动全身”，修改工作量迅速增加，最终使得软件难以维护，成为固化难改的“死”系统。这时，需要采用面向对象的程序设计方法，隐藏次要数据和实现细节，暴露清晰和简单的对外接口，软件系统才能较为容易迭代，不断添加新的功能。

4. 面向对象编程

在现实世界中，复杂的事物常常表现为对象形式，即内部有复杂的数据结构，但只需了解事物的对外接口就可以对其操作。映射到程序设计中，就是将数据和算法视为同等重要，而且将两者融合在一起，

封装之后就隐藏了内部的复杂性，抽象性得以提高[7]。基于对象的程序设计方法容易映射现实世界，如果进一步利用继承和类来实现与现实世界相符合的软件系统，可以较为自然地构造复杂系统。通过类的继承关系，子类可以方便地使用父类的操作和属性，同时也具有多态方法，使子类的行为与父类相似而又有所不同，形成所谓的面向对象的程序设计方法。

面向对象的程序设计重视对象的分析和设计，即将软件系统中独立性强的数据和操作划分出来，这与过程式程序设计强调功能的实现有所不同。面向对象的程序设计是先划分对象，再来实现功能，由于对象的独立性强、抽象性高，因此能适用于更复杂的问题，且容易随问题的变化而不断迭代，生命力更强，克服了面向过程的程序设计容易出现的固化现象[16]。

程序员在大量的面向对象程序设计实践中发现，一些对象虽然使用条件不同，但具有类似的共同特征。比如具备存储管理功能的对象，不管存储介质是内存，还是本地磁盘，抑或是云存储，都有存入和读取的基本特征。如果能在更高的抽象层次上将对象的共同属性提取出来，既可以达到代码重用的目的，使编程更加便捷，又可以使程序的健壮性得到保证，这就进一步引出了面向概念的编程方法。因此打造成熟可靠的可广泛使用的重用代码库就成为提高软件开发效率的必经途径，一些著名的类库如 STL、Boost 等逐渐形成，在现代编程中得到大量应用。

5. 面向概念编程

面向概念编程，或又可称为面向方面编程、泛型编程(Generic Programming, GP)等[17] [18]，从比面向对象更高的角度来理解世界，具有更好的抽象性，这在 C++语言中以模板来实现[19]。面向对象的特点在于抽象类型和具体的实现方式的分离，极力强调类型的一致性。而面向概念则侧重于对类型按需求进行语义分类，分类的结果是得到不同的概念(Concept)或方面(Aspect)。概念是对类的共同特征的进一步抽象，一个类可以同时属于不同的概念，一个概念可以对应到具有相似特征的多个类。

面向对象通过继承机制将子类和父类耦合，形成语义间的层次关系，这虽然便于重用相同操作，但也使对象间耦合度过高，妨碍了对象的独立性和弹性，即造成所谓的代码交织问题，后来提出了各种设计模式来试图解决问题[9]。设计模式是面向对象领域为实现高级代码复用而总结出来的各种方案，但设计模式的应用又要求增加更多的类，要进行更多的设计工作，增加了程序员的使用困难。

概念是一组能支持相应操作的类的集合，属于某一概念的类型，可称为符合该概念的对象类型，如对象值可以被复制并可被赋值，在 C++语言中就是能提供“=”操作。面向概念编程带来了更大规模的复用、更好的通用性、更高的抽象程度。另外，面向概念的泛化类一般很少使用虚函数，这在很大程度上减少了运行期的成本，把很多运行期开销转移到编译期，提高了程序运行速度，同时在编译期就能进行错误检查，提高了程序的健壮性。相对于用面向对象的继承方式来实现类的定义，面向概念可用少量的泛化组件来实现更多的类，同时保证实现的安全性。因此面向概念编程在得到极大规模的代码复用的同时，也能获得巨大的弹性。面向概念可以说是将对象的不同属性抽离出来，然后再正交组合在一起，实现既复用又安全的类体系，因此是比面向对象更为抽象的程序设计方法。

C++中的 STL (Standard Template Library)是现今最成功的面向概念编程的范例[20] [21]。其基本概念是把抽象的软件行为用标准的分类学分类，将面向对象的数据和算法的紧密封装进行了拆分，以达到更高层次的重用。STL 强调数据和算法的分离，算法不依赖于具体的数据，数据可完全独立于算法。为融合数据和算法，又提出了采用迭代器进行连接。数据包含于容器中，容器是可以包含并管理其他对象的一种对象，它提供迭代器，用以对其所包含元素进行操作。大多数的算法并不直接作用于容器上，而是操作于迭代器所形成的区间中。迭代器是 STL 的核心，迭代器将容器与作用其上的算法分离，算法通过迭代器操作数据，数据则通过迭代器为算法提供必要的操作上的支持。因此在 STL 中数据和算法先行分

离，然后通过迭代器耦合在一起，就达到了比面向对象更高层次的抽象。

6. 面向组件编程

随着信息技术渗透到社会生产生活的方方面面，软件规模越来越大，内部结构越来越复杂，需要在不断提高的抽象层次来编程，从面向过程到面向对象，再到面向概念，才能同时保证软件开发的便捷性和可靠性。现代编程语言如 C++等，能够同时支持以上编程模式，基本的语言特性支持面向过程，类支持面向对象，模板技术支持面向概念。但在更大的范围看，如 Windows 平台上，存在多种编程语言，如何在编程时重用多种语言的已有代码呢？这就需要在二进制层面建立起沟通多种编程语言的桥梁。

在 Windows 平台上，用动态链接库(DLL)可以重用不同语言编写得到的二进制代码，基本的方法是通过函数的使用方式，这可认为是面向过程的封装方法。如果封装为二进制层面的对象，可称之为组件化或构件化编程[22]-[24]。组件化程序设计使用面向对象的方法实现组件，强调真正的软件可重用性和高度的互操作性，然后可利用组件的产生和装配来构成软件系统，使软件产品采用类似于搭积木的方式快速建立起来。由于采用经过验证的组件，因此不仅可以缩短软件产品的开发周期，同时也提高了系统的稳定性和可靠性。从某种意义上说，面向组件编程就是二进制层面上的面向对象编程，是跨越了不同语言的更大范围的对象复用。

基于组件的现代软件开发模式中，软件开发人员被自然地划分为三种类型：组件生产者、组件管理者和组件复用者[1]。组件管理者提出组件的生产和使用标准，组件生产者可据此创建具有独立功能的组件，这些组件经过编译以二进制方式提供，因此是立即可用的，不需再经过漫长的重新编译阶段，而且使用的范围是灵活的，并不固定于某个软件系统。组件的复用者则进行基于组件的具体软件系统开发，包括组件查询、组件理解、组件的适应性修改、组件组装以及构成系统的演化等。

组件的定义标准是面向组件开发的核心，直接关系到组件的生产者和复用者，需要同时具有稳定性和灵活性。对象管理组织(Object Management Group, OMG)提出了公共对象请求中介体系结构(Common Object Request Breaker Architecture, CORBA)，而微软提出了组件对象模型(Component Object Model, COM)，并不断扩充发展到 COM++，成为 Windows 平台上的组件标准[13]。COM 核心定义了 COM 对象组件与其使用者如何通过二进制标准接口进行交互的规格说明，具有语言无关性、进程透明性、可重用性，不同语言开发的 COM 对象可以交互、共享。COM 提供的组件服务可以采用 DLL 形式在客户进程空间内运行，效率高，但安全性差，也可以采用 EXE 形式运行在另一个进程空间中，稳定性好，但效率相对低一些。组件对象产生于面向对象模型，有关面向对象的程序开发方法和设计技术也基本适用于组件程序设计，但二者也存在一些差别，组件的封装性更高，是二进制层面的对象模型，是独立于编程语言的对象模型，是面向对象的发展和升华。

7. 组件市场编程

在组件发展成熟的基础上，功能相似、可以互相替代的组件将会由不同公司或个人开发出来，在市场上相互竞争，这就形成了组件商店或组件市场。此时软件的开发是基于灵活开放、动态和多变的组件框架，将各类组件装配为基本系统，然后经历由“无序”到“有序”的往复循环过程，软件体系结构侧重点从基于实体的结构分解转变为基于协同的实体聚合。将组件中具有同样功能的部分提取出来再加以整合利用，这可以称为在二进制层面上与开发语言无关的面向概念编程。

随着软件运行平台从集中封闭的环境向灵活开放的互联网环境发展，组件市场也必将扩展到整个互联网，形成所谓的网构软件[1] [4]。比如随着智能化的大型语言模型的推广，其他软件也将利用这一工具升级为智能化软件，只要提供同样的接口，具体调用的组件是灵活可变的，可以是本地的快速模型，也

可以是云上的复杂模型，还可以在一个模型服务出现故障时自动切换到另一个模型，这就是利用了多个可替换组件的相同特性来构造柔性软件。网构软件对于传统软件结构的突破体现在，从系统目标的确定性到多重不确定性的转变，从系统演化的静态性到动态性的转变，从构成单元的被动性到主动自主性的转变，从协同方式的单一性到灵活多变性的转变，从经验驱动的软件手工开发模式到知识驱动的软件自动生成模式的转变[1]。

从技术角度看，面向组件市场的软件系统将在开放的环境下运行，并以多种协同方式与其他软件实体进行网络环境下的互连、互通和联盟，从本机环境的相同功能组件的灵活替换，直至整个互联网环境的组件发现、服务价格比较、区块链交易等。软件实体更加类似于个人参与市场交易的情形，既需要发现信息，又需要提供服务，软件构成部分相比以往系统更加灵活，并不断进行迭代和演化，最终程序框架和动态数据融合为一体，形成个性化的千人千面的软件智能体，这可以称为智能化的网构软件[15]。

8. 自然语言编程

自从 ChatGPT 在 2022 年 11 月底推出，这一人工智能聊天机器人在短短 5 天内，注册用户数就超过了 100 万，这标志着人工智能技术在对话生成、问答系统、文本生成、文本翻译等方面的巨大进步。通义灵码产品技术负责人陈鑫认为，编程就是最高频的 AI 应用场景，因此通过大型语言模型对人类自然语言的理解，AI 将导致程序设计方法的奇点降临，已有的成熟程序设计范式发生新一轮迭代，AI 生成工具将重构软件开发工具，进而重塑程序员的工作方式。

既然 ChatGPT 已能实现从自然语言到自然语言的翻译，那么从自然语言到计算机高级语言的翻译也可说是顺理成章、顺手拈来。本来自然语言是模糊的，编程语言是精确的，要跨越它们之间的“鸿沟”，必须要在足够智能的情况下才能实现[25] [26]。聊天机器人作为编程助手，可随时将自然语言翻译为各种计算机高级语言，不准确的地方还可以通过人机交互来修正，这样将大大降低普通人学习编程的门槛，也在很大程度上提高了程序员的编程效率。给机器下达指令将不再是程序员的专利，人人都能用自然语言去创建应用，编写软件的门槛将急剧降低，动嘴即可编程的时代到来。因此，基于自然语言的编程方法将使编程人员的数量急剧扩大，软件开发即将从阳春白雪变成下里巴人，成为人工智能时代人人皆可掌握的必备技能。

AI 技术在软件开发工具中的渗透刚开始是采用深度学习方法从成熟代码库中提取编程技巧[27]，通过 ChatGPT 查询代码知识，或使用 Copilot 自动补全代码。到 2024 年，一些开发工具更深入地整合了大型语言模型，使其拥有更完美的上下文，还能处理更大块的代码生成，在某些情况下甚至能端到端地自动执行某些工作流程。如微软公司的 GitHub Copilot 已被程序员广泛使用，其订购收入规模已经超过 GitHub 的收购价格。AI 编程初创公司 Anysphere 推出了集成开发工具 Cursor，可以用纯英文自动驱动，还能从代码库中获取最佳答案。

在程序设计的自然语言编程阶段，AI 工具将着眼于辅助人完成任务[28] [29]，还不会改变软件工程专业分工，即工具负责赋能人员提效，人负责主导、提示及确认。AI 工具将作为编程助手出现，包括将自然语言翻译为程序代码，通过交互修改程序代码，自动检查程序代码，自动生成测试用例等，需要克服的技术难点有：生成的准确度、推理性能、数据个性化和代码安全等。美国伊利洛伊大学张令明团队使用两阶段法定位和修复软件代码错误，在定位阶段，用分层方式逐步细化到可疑的文件、类/函数，直到具体的编辑位置，而在修复阶段通过生成多个候选补丁，并对其进行过滤和排序，以选出最可能的修复代码。

要提高生成准确度，AI 编程助手必须要有过硬的基础模型能力，能正确理解自然语言并翻译为对应的程序代码。同时要注意推理性能，减少辅助延迟，尽量达到即时无感的流畅状态，可通过分级缓存和

丰富的模型组合，从而实现速度与准确性兼顾。可训练小参数代码模型来完成时延敏感型的代码补全任务，使用中等参数模型提供代码解释、注释生成、单元测试、代码优化等常见代码技能，至于对模型知识面、编程能力、推理能力有更高要求的研发问答，则需要最大参数模型并叠加互联网实时检索增强生成(RAG)技术，消除模型幻觉，提升回答质量。

AI 编程助手能让开发者的效率变得更高，从而能让开发者成为超级程序员，把更多的时间和精力花在思考和创造上。如百度公司的文心快码(Baidu Comate)是基于文心大模型的智能代码助手，截至 2023 年 12 月，文心快码已覆盖 100+种编程语言，支持 10+种主流开发工具平台，为软件研发全生命周期提供全场景智能辅助。在百度文心快码的帮助下，“软件开发”的含义不再是孤独地敲打键盘，而是用简单的自然语言表述出一个解决方案，然后在文心快码的辅助下把程序代码自动创建出来。目前多数情况下，AI 在编程中的角色还只是辅助，但由 AI 驱动编程的未来还将进一步发展，已经可以看到一些苗头，将继续向自主进化编程方向发展。

9. 自主进化编程

从 2024 年 3 月首个 AI 全自动软件工程师 Devin 提出以来，针对软件工程的 AI 智能体的设计成为研究前沿。编码智能体可以编写代码，然后运行程序；或者编写代码，编写测试，并检查一致性。这种类型的自动监督在大多数领域是不可能实现的，而在程序设计领域则较容易达到。编码智能体将不断提升推理和规划能力，能够自学新技术，自主查找并修复代码缺陷。因此，软件工程很快将随着 AI 的快速进步发生根本性的变化，真正的编码智能体将能够完成端到端的任务，并与今天的编程助手相辅相成，成为真正的“虚拟程序员”，代替人类程序员完成各类研发工作，使程序员从繁琐的开发任务中解放出来。

真正的编码智能体将是一个可以在自主模式和同步模式之间无缝切换的智能系统。自主模式适用于范围明确的工作，智能体将完全独立工作，具备编写和运行代码、使用外部工具、搜索网络信息、访问内部文档以及从过去错误中学习的能力。它会不断迭代任务，直到完成或遇到瓶颈，这将占到 80%的工作量。智能体还可为其工作中的简单部分派遣更便宜的子智能体，从而降低计算成本和延迟。同步模式可称之为协同模式，或者配对编程模式，这适用于最难的任务。人类程序员将在高层次上指导编码智能体，而编码智能体负责处理低层次的实现细节。互动将是高度多模态的，人类和智能体将在文本描述、视觉图表、口头讨论和直接操作彼此代码之间无缝切换。通过共享屏幕，智能体跟随程序员并给出建议和意见，或者智能体共享它的屏幕，而程序员在旁边给予指导，整个互动过程就如同项目开发经理和下属程序员的交流一样。

正如计算器和电子表格这样的工具并没有让会计师失业，编码智能体在降低编写软件门槛的同时，人类程序员仍然需要在更高层次上工作。每个程序员都将成为项目经理，并配有由数十个编码智能体组成的工作小组。程序员只需将基本任务委派给编码智能体，然后就能把更多的时间花费在解决更高层次的问题上：理解需求，架构系统以及决定构建什么。此时，编码智能体就从编程助手进化为数字员工，能够切切实实地自动完成分派的简单任务。从外界观察，应用程序能够在数字员工的操作下不断发生更改以适应需求，仿佛在进行自主进化。人类程序员的工作量将越来越少，工作内容将越来越高层抽象。

程序员的终极梦想就是编写一款程序来代替自己编程，到真正实现的时候才发现这既是一种幸福，也是一种悲哀。有了数字员工，几乎没有什么代码还需要程序员手工编写，程序员这一角色也将逐渐隐退，只有架构师、项目经理等职位还暂时得以保留。因此可以说，人类程序员是人工智能时代加速推进的前锋，也有可能是首当其冲被机器替代的工种，大量的从事低阶工作的程序员将面临失业。当然，最理想的状态是在人机协同的情况下，程序员都变身为项目经理，工作效率越来越高，工作时间越来越短，而我们将进入一个前所未有的软件繁荣时代，各类应用软件将以比以往成百上千倍的数量涌现出来。

随着人工智能技术的进一步发展，编码智能体学习人类的编程经验将越来越纯熟，在与程序员的互动中越来越个性化，直至几乎可以替代程序员本身。如果从编程技能逐步扩展到其他方面，智能体将越来越了解人类个体，甚至可以在多个场合代替个人自动做出决策，最终成为个人的“数字替身”。在数字世界中，该替身还可不断学习、进化，并与其他智能体协作，完成各种不同任务，同时也会获得对应的报酬。数字替身的能力越强，获得的报酬越多，个人更可能不断训练它，使之在行为上越来越接近人类个体。那么，在个体生命结束后，该数字替身还可以继续“工作”，成为人类社会的“虚拟人口”，继续为社会创造价值。从某种意义上来说，人类个体通过这种方式达到了“数字永生”，一大批社会精英将因此而永久留存专家经验，持续不断地在数字世界服务大众，成为社会生产力的不可忽视的一部分。

10. 结语

计算机信息技术的发展主要表现为软件，各类应用都必须通过软件来实现。随着软件功能越来越复杂，规模越来越大，软件开发也逐渐成为需要团队协作的系统工程，需要越来越先进的技术手段来支持，这使得程序设计方法得到迅猛改进和更迭。本文从软件开发的便捷性的进步，也即从软件设计的抽象性的提高来考察程序设计范式的演进方向。从直接使用底层机器指令的汇编语言，到发展到面向过程的计算机高级语言，再到面向对象、面向概念的设计范式，软件开发的抽象程度越来越高，但开发效率也与日俱增。在选择具体项目的程序设计范式时，需要在程序运行效率和开发效率、程序的时间需求和空间需求、程序的稳固性和灵活性、代码的简单性和可重用性等这些内在矛盾之间作出艰难的取舍，并没有万能的灵丹妙药，必须针对应用问题本身选择适用的类型，并可能在后续迭代中发生改变。总体来看，已有的稳定程序库，不管是源代码库还是二进制组件库，都提供了经过多重考核的基础构件，只要善加利用就可在更高层次上开发出可靠运作的应用程序，但这也增加了程序员的学习负担，减慢了程序员的成长速度，导致高级开发人员难以满足实际需求。

人工智能技术的发展对解决应用需求和人才供给之间的矛盾提供了可能的解决方案，随着大型语言模型的应用，计算机能够较准确领会自然语言描述的语境，并将之翻译为精确化的计算机高级语言，充当了自然语言转换到程序代码的媒介，因此编程将从面向计算机高级语言转化为基于自然语言。隔离了编程语言的底层差异后，很多时候无需再专门学习不同的编程语言，人人都可直接使用自然语言进行编程，基于自然语言的程序设计方法将方兴未艾。随着智能技术的进一步提升，编码智能体还将从智能助手升级为数字员工，每个程序员都将升级为超级个体，带领众多的数字员工，迎来应用软件开发的井喷时代。若智能技术继续飞跃，数字员工将进一步进化，还可能成为程序员的“数字替身”，直至发展到“数字永生”，彼时“虚拟程序员”将无时无刻地以大师级水准在数字世界进行程序架构、编码、测试、迭代。人类程序员在成为创造数字世界的“上帝”后，潇洒地挥一挥手，然后可以一劳永逸地退居幕后。

参考文献

[1] 杨芙清. 软件工程技术发展思索[J]. 软件学报, 2005, 16(1): 1-7.

[2] 杨芙清. 从软件工程入手[J]. 软件世界, 2005(1): 40-41.

[3] 梅宏, 申峻嵘. 软件体系结构研究进展[J]. 软件学报, 2006, 17(6): 1257-1275.

[4] 梅宏, 黄罡, 刘譞哲, 等. 网构软件研究: 回顾与展望[J]. 科学通报, 2022, 67(32): 3780-3792.

[5] 赵永升, 宋丽华. 程序设计方法 SPP 与 OOP 的比较[J]. 烟台师范学院学报(自然科学版), 2002, 18(3): 229-233.

[6] 刘琴. 软件工程中程序设计方法的比较[J]. 计算机时代, 2017(7): 53-55.

[7] Lippman, S.B. 深度探索 C++对象模型[M]. 侯捷, 译. 武汉: 华中科技大学出版社, 2001.

[8] 皮德常, 王宁生. 一种基于可扩展对象的程序设计方法[J]. 计算机应用研究, 2002(1): 113-114, 120.

[9] 陈叶旺, 余金山. 泛型编程与设计模式[J]. 计算机科学, 2006, 33(4): 253-257.
[10] 孙斌. 面向对象、泛型程序设计与类型约束检查[J]. 计算机学报. 2004, 27(11): 1492-1504.
[11] 潘爱民. COM 原理与应用[M]. 北京: 清华大学出版社, 1999.
[12] 李志辉, 崔洪芳. 组件化程序设计方法和面向对象程序设计方法的比较[J]. 福建电脑, 2004(11): 21, 11.
[13] 李杰, 韩建敏. 组件化程序设计方法的探讨[J]. 福建电脑, 2006(4): 66-67.
[14] 黄罡, 梅宏, 杨芙清. 基于反射式软件中间件的运行时软件体系结构[J]. 中国科学: 技术科学, 2004, 34(2): 121-138.
[15] 张伟, 梅宏. 基于互联网群体智能的软件开发: 可行性、现状与挑战[J]. 中国科学: 信息科学, 2017, 47(12): 1601-1622.
[16] Koenig, A., Moo, B. C++沉思录[M]. 黄晓春, 译. 北京: 人民邮电出版社, 2008.
[17] 古思山, 蔡树彬, 李师贤. 从面向方面程序设计的定义到面向方面程序设计语言[J]. 计算机科学, 2011, 38(10): 133-139.
[18] 左正康, 刘志豪, 黄箐, 等. Apla 与程序设计语言泛型特性比较研究[J]. 江西师范大学学报(自然科学版), 2019, 43(5): 454-461.
[19] Vandevoorde, D., Josuttis, N.M. C++ Templates 中文版[M]. 陈伟柱, 译. 北京: 人民邮电出版社, 2004.
[20] Plauger, P.J., 等. C++ STL 中文版[M]. 王昕, 译. 北京: 中国电力出版社, 2002.
[21] 侯捷. STL 源码剖析[M]. 武汉: 华中科技大学出版社, 2002.
[22] 徐敏, 周定康. 组件技术在软件开发中的应用[J]. 计算机与现代化, 2002(2): 1-3.
[23] 徐春晓, 孙涌. 一种嵌入式轻量级 GUI 构件的实现方法[J]. 苏州大学学报(工科版), 2009, 29(3): 1-4.
[24] 茅正冲, 叶臻, 黄芳. 基于构件的可配置嵌入式应用程序设计模式[J]. 计算机测量与控制, 2015, 23(4): 1432-1434, 1437.
[25] 高寅生, 扬志远. 跨越自然语言与编程语言的“鸿沟” [J]. 云南民族学院学报(自然科学版), 2002, 11(1): 552-555.
[26] 左燕. 自然语言与计算机编程语言的相通性微探[J]. 现代信息科技, 2019, 3(22): 28-29.
[27] 刘晋宇. 基于深度学习的自然语言编程任务分解研究[D]: [硕士学位论文]. 长沙: 国防科技大学, 2018.
[28] 马骋原, 李晓辉, 汪涵. 面向智能合约的可编程自然语言设计[J]. 电子设计工程, 2020, 28(4): 10-15, 22.
[29] 王羽, 葛唯益, 李青山, 等. 基于自然语言的跨编程语言微服务集成方法研究[C]//中国指挥与控制学会. 第九届指挥控制大会论文集. 2021: 354-358.