低代码的现状与目标分析

接上一篇《谈谈低代码与AI大模型》，本篇主要细化并阐述一下为什么需要低代码平台、经过多年发展后低代码平台面临的困局以及如何破局的一些个人想法。

1. 低代码并不是新东西

虽然业界一直在说低代码，低代码，其实低代码很好理解，按照字面意思就可以分析出个大概，低代码开发平台（LCDP）是无需编码（零代码）或通过少量代码就可以快速生成应用程序的开发平台。

早在上世纪八十年代，就有美国公司和实验室开始研究程序可视化编程这一领域，做出了4GL“第四代编程语言”，后来2000年衍生成VPL（Visual Programming Language可视化编程语言）。到2014年，Forrester Research正式提出了“低代码/零代码”的概念，定义是“利用很少或几乎不需要写代码就可以快速开发应用，并可以快速配置和部署的一种技术和工具”。随后Forrester又定义了低代码开发平台，其是一种应用程序开发的平台，采用可视化、声明性的技术，而不是编程，以满足各种应用和流程自动化的使用情况。

低代码本身并不是高深的技术，其本质是通过对常用的技术组件、代码逻辑、功能组件以及开发场景的抽象，底层采用组件封装、批处理生成，前端通过界面化、拖拉拽等方式/手段，降低使用人员在开发语言、技术、框架等方面的学习及使用成本，并提高开发效率的开发平台/工具。

2. 为什么需要低代码

目前国内以及国际复杂经济大环境下，经济形势严峻，失业、消费降级给各行各业的企业带来了很大的生存压力，各企业都面临着如何提高营收以及降本增效的问题。

随着行业内甚至跨行业竞争的加剧，为提高企业自身的竞争力，企业对数据字化转型的诉求也越来越强烈，数字化转型的需求也越来越深入。

对甲方企业来说，需要在较低成本投入的情况下，能够快速的实现数字化转型的需求；对乙方软件企业来说，项目需求越来越复杂、深入，但项目的预算却逐年下降、自身的人力成本逐年提高。原来的堆人模式已经很难满足当下的竞争需求，如何做到降本增效已经是各软件企业不得不面临的问题。

低代码在降本增效方面是个非常有竞争力的可选项：

- 降低开发成本

- ✓降低学习成本：低代码通过对常用技术组件的封装，并通过界面化、拖拉拽等方式/手段，可以有效降低使用人员在开发语言、技术、框架等方面的学习及使用成本。
- ✓降低人力成本：常规普通功能通过低代码以较低的人力成本快速实现；复杂或个性化程度较高的功能，由人力成本较高的中高级技术人员定制化实现。
- ✓降低沟通成本：产品人员可以通过低代码实现demo来达到与开发人员之间更有效的沟通；甚至较简单的功能可以由产品人员直接实现。

- 增效
  - ✓提高开发效率：由于低代码平台图形化拖拉拽的开发模式以及模板化的典型功能开发能力，通过低代码能够极大的提高开发效率。
  - ✓提高测试效率：由于所见即所得的开发模式以及便捷的辅助测试能力，相比传统的测试，在效率上会有较大的提升。
- 提质
  - ✓提高版本质量：相比定制化开发的功能，通过低代码开发出来的功能会有更高的质量。

所以在未来的软件开发企业中，低代码很可能会承担起降本增效、提高竞争力的关键作用。

3. 低代码平台的困局

虽然大家都知道未来低代码的作用可能会很关键，很多企业对低代码的关注度也很高，前几年低代码也火过一阵，大家一窝蜂的建设，涌现了一堆各类的低代码平台：

- 通用领域、专业领域的各类低代码
- 前端、后端、报表、数据可视化的低代码.....
- 表单驱动、模型驱动、流程驱动的低代码.....

但是风口过后，留下的却是一地鸡毛，低代码平台的推广和应用都不是那么理想：

- 大企业不愿意用、小企业用不起
- 技术人员不愿意用、业务人员不会用
- 各大低代码厂商都活得不是那么理想

大家对低代码的吐槽也很多：

- 此类平台通常易用性极高，拓展性极低。
- 导致业务方特别容易被低代码厂商绑架。
- 国内的低代码平台扩展性很差，
- 全是黑盒，调试非常困难不说，还经常有莫名其妙的各类问题。
- 首先，现在市面上的低代码开发，只适用于后台管理类项目的制作！
- 现在的低代码平台确实还就是那么个状态，大事儿担不了，小事做不好。
- 低代码，实际上对代码的二次封装，而这次封装也重新带来了学习成本，出问题的时候调试成本也增加
- 低代码本身来说，就是解决大家一时紧需而已
- 不好做版本管理，不好diff，不好追踪当时逻辑为什么这些写，逻辑复杂后难以维护
- 只能忽悠不懂程序，又想入这一行的小白
- ……

4. 如何破局?

首先，要认清低代码的现实和目标：

- **低代码解决不了所有的问题**：低代码擅长解决一些常规的、相对通用的场景（尤其是一些重复性的场景）；对于逻辑复杂度较高或者个性化程度较高的场景，低代码并不擅长。所以在设计一个低代码平台的时候，一定要提前考虑好如何与传统编码方式融合共存的情况。
- **很难覆盖所有的开发领域**：低代码的赛道很多，包括各类技术领域和业务领域，一个低代码平台不可能把所有的赛道都覆盖，那样只会造成样样都能做一点，但都不精的情况。所以一定要选定一个或少数几个点，做精做专。
- **低代码平台不能成为一个玩具**：很多低代码平台在演示的时候都拿一些很简单的场景，通过低代码都能非常快速的实现，看起来非常美好。但是在实际的应用场景中，这样是远远不够的。低代码除了能够快速实现一些简单的、通用场景之外，还必须要能支持一些常规的复杂能力的实现，也就是说低代码需要满足常规开发场景中6、70%的需要。
- **低代码开发不能是一锤子买卖**：有不少低代码采用生成代码的模式，个人觉得这种方式是不可取的。生成代码后必然会造成后期人工进行修改的情况，这就造成了后续无法再通过低代码进行维护、升级的问题，这种属于代码生成器，而不是低代码。通过低代码开发的功能必须要具备迭代升级的能力。当然低代码本身也需要具备迭代升级的能力。
- **一家做不好低代码平台，需要形成一个生态**：当一家企业或者一个产品/项目，在开始的时候往往会先进行技术选型，如果一个低代码平台是完全私有和封闭的生态，那么选型专家们首先想到的就是后续技术依赖和被平台绑架的问题，尤其对于低代码这种将会在底层被深度使用的技术平台，这个问题会更严重。因此低代码平台最合适的建设方式应该是采用开源的方式，大家一起来建设并能形成一个合理的生态。

其次，需要考虑如何与众多同类平台竞争，做出自己的特色并形成产品优势：

- 选择一个赛道，做精做实：
  - ✓低代码不是个玩具，是需要能够解决实际问题的
  - ✓需要有丰富系统建设经验、开发经验、业务经验以及有一定前瞻性的设计/研发团队
- 形成产品优势：
  - ✓先进且合理的设计思想/理念
  - ✓明确且合理的功能规划及建设目标
  - ✓从实际应用场景从抽象并建设平台能力，解决实际问题
  - ✓既能够支撑简单/典型功能的快速实现，也能够满足较复杂业务场景的开发需求
  - ✓建设开放的平台生态，而不是封闭的平台产品
  - ✓平台本身以及基于平台开发的系统都具备迭代升级能力
  - ✓灵活的交付方式：源码交付、产品交付以及项目交付
  - ✓测试能力、调试能力、版本管理能力、协同开发能力
- 拥抱AI：
  - ✓集成AI能力，使AI能力能够更无缝的结合到业务系统的方方面面
  - ✓将AI能力深度应用到低代码平台，使低代码平台本身能够更加的智能

再次，应对企业和技术人员两个直接使用群体的担忧：

- 企业，尤其是大企业，在采购低代码平台时，主要的担忧有：
  - ✓担心被低代码平台厂商技术绑架
  - ✓平台功能不完善、集成/升级困难
  - ✓平台厂商技术支持问题
- 应对
  - ✓建议开放的平台生态，而不是封闭的平台产品
  - ✓灵活的交付方式：源码交付、产品交付以及项目交付
  - ✓可见的降本增效提质的效果
  - ✓能够解决实际问题，应对复杂业务场景
  - ✓业务与技术的无缝衔接

- 技术人员主要的担忧有：
  - ✓对自己的成长不利，容易被取代；
  - ✓换平台或公司后，前期的积累没有任何作用
  - ✓调试及问题排查困难
- 应对
  - ✓建议开放的平台生态，而不是封闭的平台产品，技术人员能够直接参与到平台的生态中，学习到各种先进的设计以及最佳实践
  - ✓平台并不解决所有问题，反而可以使技术人员从繁杂的重复工作中解脱出来，专注于过更有价值的业务以及复杂场景的开发
  - ✓完善的测试能力及调试能力
  - ✓精心设计的扩展能力

5. 其他

后续将通过更多的文章，逐步对低代码平台的设计理念、思想以及具体的设计进行细化，随着设计与开发的进展，低代码平台的源码将开源到GitHub：https://github.com/zjyzju/zPaaS-lowcode，敬请关注。

如果有低代码平台以及AI相关的讨论，可以发送邮件到我的邮箱：zjyzju@163.com 。