IBM Rational

# 迭代化软件开发技术
# IBM Rational 技术白皮书

版本 1.0

Rational software

IBM

# 目录

Rational. software

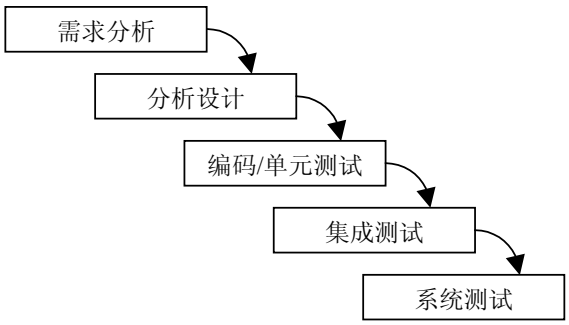
IBM

# 迭代化开发技术白皮书

## 1. 传统开发流程的问题

传统的软件开发流程是一个文档驱动的流程，它将整个软件开发过程划分为顺序相接的几个阶段，每个阶段都必需完成全部规定的任务（文档）后才能够进入下一个阶段。如必须完成全部的系统需求规格说明书之后才能够进入概要设计阶段，编码必需在系统设计完成之后才能够进行。这就意味着只有当所有的系统模块全部开发完成之后，我们才进行系统集成，对于一个由上百个模块组的复杂系统来说，这是一个非常艰巨而漫长的工作。

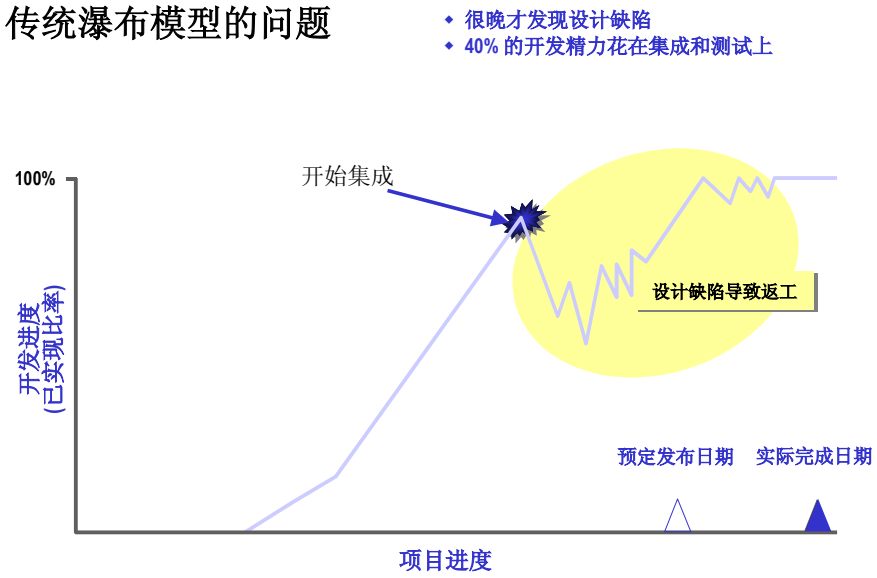
**传统的开发流程 —— 瀑布模型**




随着我们所开发的软件项目越来越复杂，传统的瀑布型开发流程不断地暴露出以下问题：

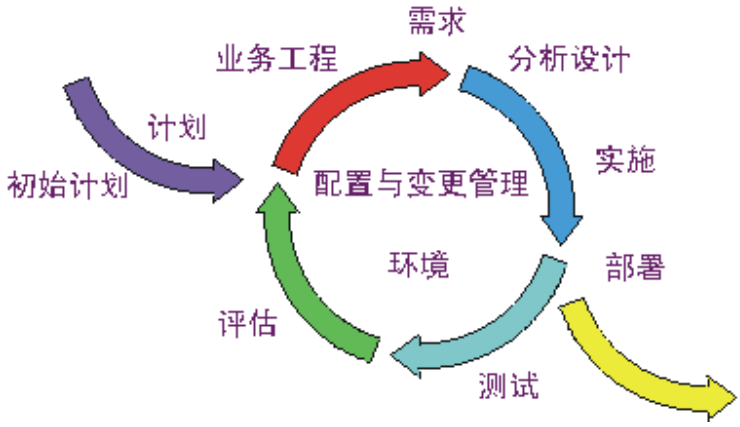
- 需求或设计中的错误往往只有到了项目后期才能够被发现
  例如：系统交付客户之后才发现原先对于需求的理解是错误的，系统设计中的问题要到测试阶段才能被发现。
- 对于项目风险的控制能力较弱
  项目风险在项目开发较晚的时候才能够真正降低，往往是经过系统测试之后，才能确定该设计是否能够真正满足系统需求。
- 软件项目常常延期完成或开发费用超出预算
  项目开发进度往往会被意外发生的问题所打乱，需要进行返工或其他一些额外的开发周期，造成项目延期或费用超支。
- 项目管理人员专注于文档的完成和审核来估计项目的进展情况
  所以项目经理对于项目状态的估计往往是不准确的，当他回答系统已完成了 80%的开发任务时，剩下 20%的开发任务实际上消耗的是整个项目 80%的开发资源。

在传统的瀑布模型中，需求和设计中的问题是无法在项目开发的前期被检测出来的，只有当第一次系统集成时，这些设计缺陷才会在测试中暴露出来，从而导致一系列的返工：重新设计、编码、测试，进而导致项目的延期和开发成本的上升。




## 2. 采用迭代化开发控制项目风险

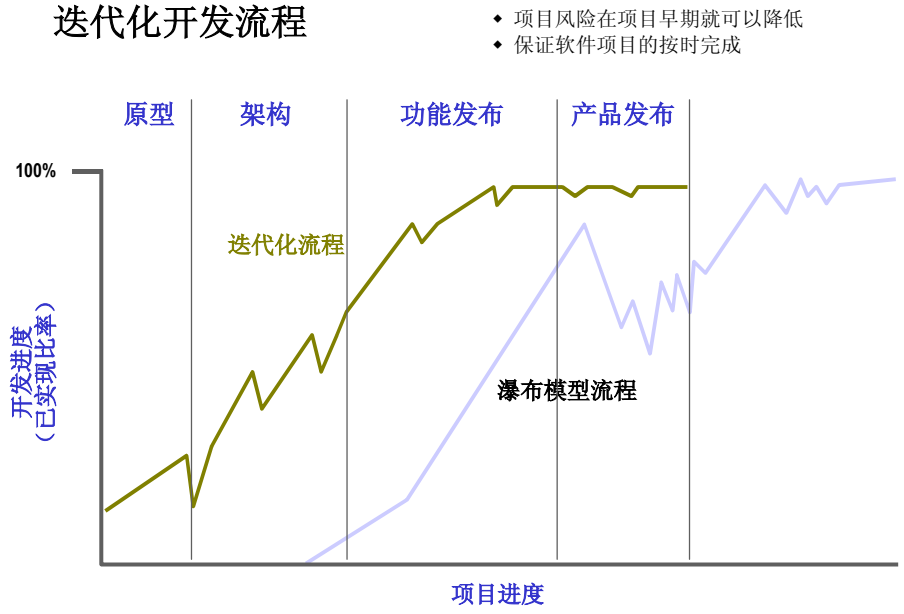
为了解决传统软件开发流程中的问题，我们建议采用迭代化的开发方法来取代瀑布模型。在瀑布模型中，我们要完成的是整个软件系统开发这个大目标。在迭代化的方法中，我们将整个项目的开发目标划分成为一些更易于完成和达到的阶段性小目标，这些小目标都有一个定义明确的阶段性评估标准。迭代就是为了完成一定的阶段性目标而所从事的一系列开发活动，在每个迭代开始前都要根据项目当前的状态和所要达到的阶段性目标制定迭代计划，整个迭代过程包含了需求、设计、实施（编码）、部署、测试等各种类型的开发活动，迭代完成之后需要对迭代完成的结果进行评估，并以此为依据来制定下一次迭代的目标。



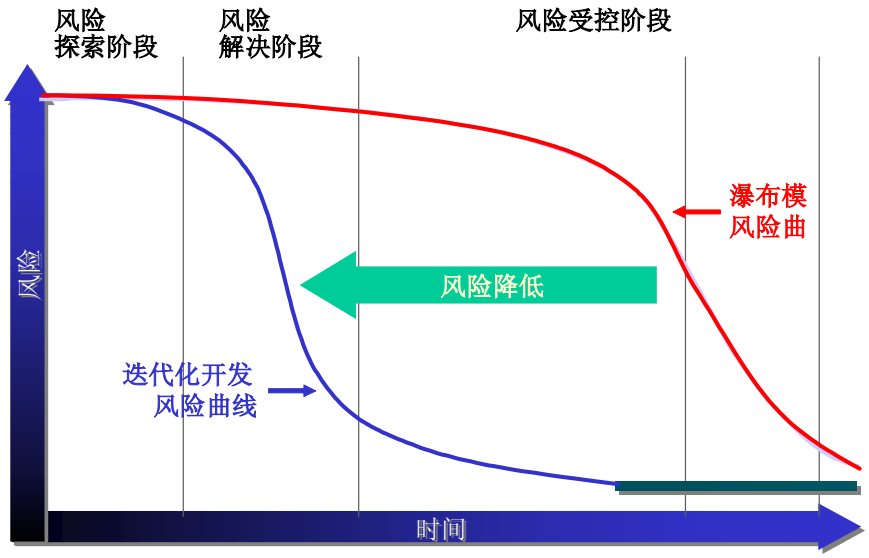

与传统的瀑布式开发模型相比较，迭代化开发具有以下特点：

- 允许变更需求
  需求总是会变化，这是事实。给项目带来麻烦的常常主要是需求变化和需求“蠕变”，它们会导致延期交付、工期延误、客户不满意、开发人员受挫。通过向用户演示迭代所产生的部分系统功能，我们可以尽早地收集用户对于系统的反馈，及时改正对于用户需求的理解偏差，从而保证开发出来的系统真正地解决客户的问题。
- 逐步集成元素
  在传统的项目开发中，由于要求一下子集成系统中所有的模块，集成阶段往往要占到整个项目很大比例的工作量（最高可达 40%），这一阶段的工作经常是不确定并且非常棘手。在迭代式方法中，集成可以说是连续不断的，每一次迭代都会增量式集成一些新的系统功能，要集成的元素都比过去少得多，所以工作量和难度都是比较低的。
- 尽早降低风险
  迭代化开发的主要指导原则就是以架构为中心，在早期的迭代中所要解决的主要问题就是尽快确定系统架构，通过几次迭代来尽快地设计出能够满足核心需求的系统架构，这样可以迅速降低整个项目的风险。等到系统架构稳定之后，项目的风险就比较低了，这个时候再去实现系统中尚未完成的功能，进而完成整个项目。
- 有助于提高团队的士气
  开发人员通过每次迭代都可以在短期内看到自己的工作成果，从而有助于他们增强信心，更好地完成开发任务。而在非迭代式开发中，开发人员只有在项目接近尾声时才能看到开发的结果，在此之前的相当长时间，大家还是在不确定性中摸索前近。

- 生成更高质量的产品
  每次迭代都会产生一个可运行的系统，通过对这个可运行系统进行测试，我们在早期的迭代中就可以及时发现缺陷并改正，性能上的瓶颈也可以尽早发现并处理。因为在每次迭代中总是不断地纠正错误，我们可以得到更高质量的产品。
- 保证项目开发进度
  每次迭代结束时都会进行评估，来判断该次迭代有没有达到预定的目标。项目经理可以很清楚地知道有哪些需求已经实现了，并且比较准确地估计项目的状态，对项目的开发进度进行必要的调整，保证项目按时完成。
- 容许产品进行战术改变
  迭代化的开发具有更大的灵活性，在迭代过程中可以随时根据业务情况或市场环境来对产品的开发进行调整。例如为了同现有的同类产品竞争，可以决定采用抢先竞争对手一步的方法，提前发布一个功能简化的产品。
- 迭代流程自身可在进行过程中得到改进和精炼
  一次迭代结束时的评估不仅要从产品和进度的角度来考察项目的情况，而且还要分析组织和流程本身有什么待改进之处，以便在下次迭代中更好地完成任务。



迭代化方法解决的主要是对于风险的控制问题，从下图可以看出，传统的开发流程中系统的风险要到项目开发的后期（主要是测试阶段）才能够被真正降低。而迭代化开发中的风险，可以在项目开发的早期通过几次迭代来尽快地解决掉。在早期的迭代中一旦遇到问题，如某一个迭代没有完成预定的目标，我们还可以及时调整开发进度以保证项目按时完成。一般到了项目开发的后期（风险受控阶段），由于大部分高风险的因素（如需求、架构、性能等）都已经解决，这时候只需要投入更多的资源去实现剩余的需求即可。这个阶段的项目开发具有很强的可控性，从而保证我们按时交付一个高质量的软件系统。



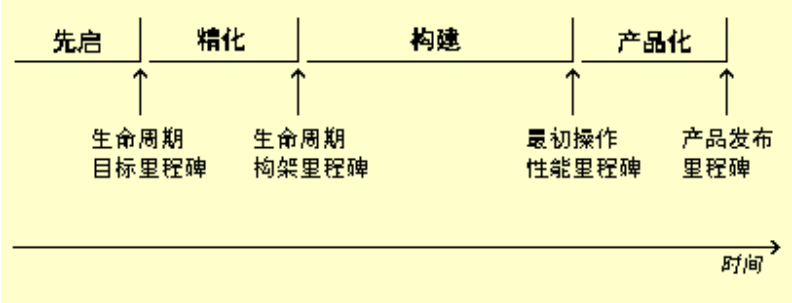

迭代化开发不是一种高深的软件工程理论，它提供了一种控制项目风险的非常有效的机制。在日常的工作我们也经常地应用到这一基本思想，如对于一个非常大型的工程项目，我们经常会把它分为几期来分步实施，从而把复杂的问题分解为相对容易解决的小问题，并且能够在较短周期内看到部分系统实现的效果，通过尽早暴露问题来帮助我们及早调整我们的开发资源，加强项目进度的可控程度，保证项目的按时完成。

## 3. 管理迭代化的软件项目

当我们在实际工作中实践迭代化思想时，Rational 统一开发流程 RUP(Rational Unified Process)可以给予我们实践的指导。RUP 是一个通用的软件流程框架，它是一个以架构为中心、用例驱动的迭代化软件开发流程。RUP 是从几千个软件项目的实践经验中总结出来的，对于实际的项目具有很强的指导意义，是软件开发行业事实上的行业标准。

## 3.1 软件开发的四个阶段

在 RUP 中，我们把软件开发生命周期划分为四个阶段，每个阶段的结束标志就是一个主要的里程碑（如下图所示）。




这四个阶段主要是为了达到以下阶段性的目标里程碑:

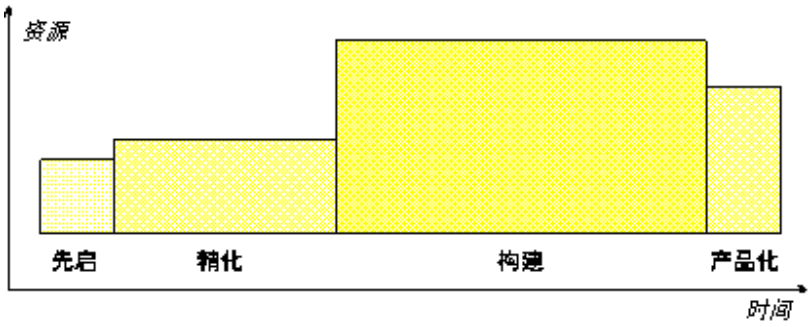
- 先启(Inception): 确定项目开发的目标和范围
- 精化(Elaboration): 确定系统架构和明确需求
- 构建(Construction): 实现剩余的系统功能
- 产品化(Transition): 完成软件的产品化工作，将系统移交给客户

每个目标里程碑都是一个商业上的决策点，如先启阶段结束之后，我们就要决定这个项目是否可行、是否要继续做这个项目。每一个阶段都是由里程碑来决定的，判断一个阶段是否结束的标志就是看项目当前的状态是否满足里程碑中所规定的条件。

从这种阶段划分模式中可以看出，项目的主要风险集中在前两个阶段。在精化阶段中经过几次迭代后，我们要为系统建立一个稳定的架构，在此之后再实现更多的系统需求时，不再需要对该架构进行修改。同时，在精化阶段中，我们通过迭代来不断地收集用户的需求反馈，便得系统的需求逐步地明确和完整。

## 3.2 关于开发资源的分配

基于 RUP 风险驱动的迭代化开发模式，我们只需要在项目的先启阶段投入少量的资源，对项目的开发前景和商业可行性进行一些探索性的研究。在精化阶段再投入多一些的研发力量来实现一些与架构相关的核心需求，逐步地把系统架构搭建起来。等到这两个阶段结束之后，项目的一些主要风险和问题也得到了解决，这时候再投入整个团队进行全面的系统开发。等到产品化阶段，主要的开发任务已经全部完成，项目不再需要维持一个大规模的开发团队，开发资源也可以随之而减少。在项目开发周期中，开发资源的分配可以如下图所示。




这样安排可以最充分有效地利用公司的开发资源，缓解软件公司对于人力资源不断增长的需求，从而降低成本。另外一方面，由于前两个阶段（先启和精化）的风险较高，我们只是投入部分的资源，一旦发生返工或是项目目标的改变，我们也可以将资源浪费降到最低点。在传统的软件开发流程中，对于开发资源的分配基本上是贯穿整个项目周期而不变的，资源往往没有得到充分有效地利用。

基于这种资源分配模式，一个典型的项目在项目进度和所完成的工作量之间的关系可能如下表中的数据所示。

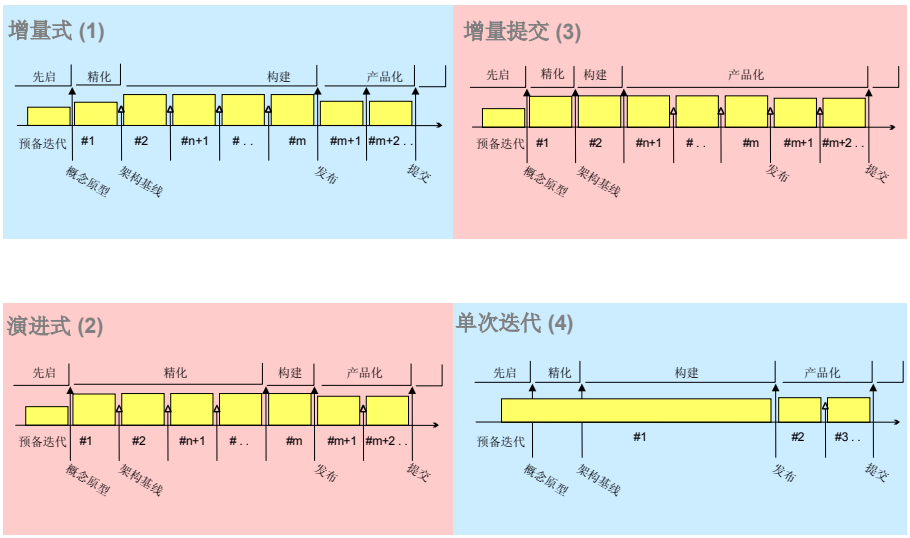
| | 先启 | 精化 | 构建 | 产品化 |
|---|---|---|---|---|
| 工作量 | ~5% | 20% | 65% | 10% |
| 进度 | 10% | 30% | 50% | 10% |

## 3.3 迭代策略

关于迭代计划的安排，通常有以下四种典型的策略模式:

- 增量式(Incremental)
  这种模式的特点是项目架构的风险较小（往往是开发一些重复性的项目），所以精化阶段只需要一个迭代。但项目的开发工作量较大，构建阶段需要有多次迭代来实现，每次迭代都在上一次迭代的基础上增加实现一部分的系统功能，通过迭代的进行而逐步实现整个系统的功能。
- 演进式(Evolutionary)
  当项目架构的风险较大时（从未开发过类似项目），需要在精化阶段通过多次迭代来建立系统的架构，架构是通过多次迭代的探索，逐步演化而来的。当架构建立时，往往系统的功能也已经基本实现，所以构建阶段只需要一次迭代。
- 增量提交(Incremental Delivery)
  这种模式的特点产品化阶段的迭代较多，比较常见的例子是项目的难度并不大，但业务需求在不断地发生变化，所以需要通过迭代来不断地部署完成的系统；但同时又要不断地收集用户的反馈来完善系统需求，并通过后续的迭代来补充实现这些需求。应用这种策略时要求系统架构非常稳定，能够适应满足后续需求变化的要求。

- 单次迭代(Grand Design)
  传统的瀑布模型可以看作是迭代化开发的一个特例，整个开发流程只有一次迭代。但这种模式有一个固有的弱点，由于它对风险的控制能力较差，往往会在产品化阶段产生一些额外的迭代，造成项目的延误。




这几种迭代策略只是一些典型模式的代表，实际应用中应根据实际情况灵活应用，最常见的迭代计划往往是这几种模式的组合。

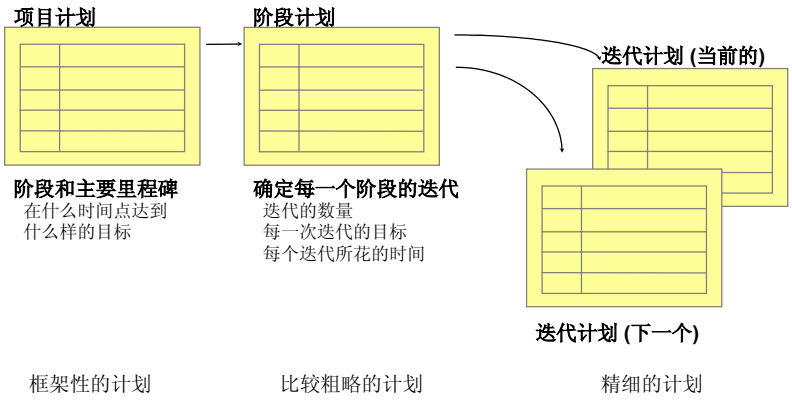
## 3.4 制定项目开发计划

在迭代化的开发模式中，项目开发计划也是随着项目的进展而不断细化、调整并完善的。传统的项目开发计划是在项目早期制定的，项目经理总是试图在项目的一开始就制定一个非常详细完善的开发计划。与之相反，迭代开发模式认为在项目早期只需要制定一个比较粗略的开发计划，因为随着项目的进展，项目的状态在不断地发生变化，项目经理需要随时根据迭代的结果来对项目计划进行调整，并制定下一次迭代的详细计划。

在 RUP 中，我们把项目开发计划分为以下三部分:

- 项目计划
  确定整个项目的开发目标和进度安排，包括每一个阶段的起止时间段。
- 阶段计划
  当前阶段中包含有几个迭代，每一次迭代要达到的目标以及进度安排。

- 迭代计划
  针对当前迭代的详细开发计划，包括开发活动以及相关资源的分配。




项目开发计划也是完全体现迭代化的思想，每次迭代中项目经理都会根据项目情况来不断地调整和细化项目开发计划。迭代计划是在对上一次迭代结果进行评估的基础上制定的，如果上一次迭代达到了预定的目标，那么当前迭代只需要解决剩下的问题；如果上一次迭代中留有一些问题还没有解决，则当前迭代还需要继续去解决这些问题。所以必须注意，迭代是不能重叠的，即你还没有完成当前迭代时，你决不能进入下一迭代，因为下一次迭代的计划是根据当前迭代的结果而制定的。