开发模型的演化

2020年8月20日 by 高玉山 — 1 Comment

> “ 其实所有的开发模型都是为了解决一个问题：如何将需求变成软件。

最开始人们心目中的过程应该是这样的：

如何将需求变成软件

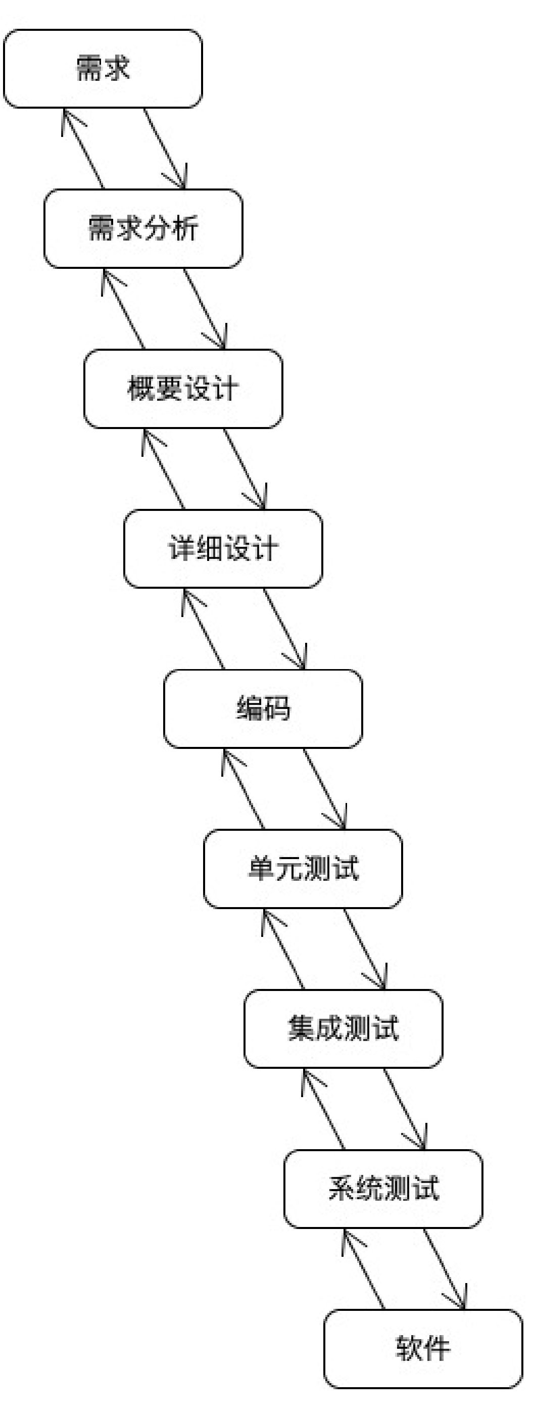
中间的“开发过程”开始很简单，只是硬件的一个配置。但这个配置越来越多，当这个配置多到难以理解时，就变成魔法了。这个魔法威力强大，但是神秘，且不可控。人们面对旺盛的需求，渴望解开魔法的面纱，将它牢牢控制在手心。于是各种开发模式纷纷登场。

瀑布模型

瀑布模型是一个经典模型，不用废话，它一定在你心里。

- 它通过里程碑，将大的项目变成小的、可控的工作；
- 它通过里程碑的**环环相扣，顺序操作**，让方案简单可以实施

通过这套流程，做到了——我的成功可以复制。（至少比魔法好）



(Waterfall Model)

各个环节都通过文档衔接，这样每个环节就可以独立了。但很快人们发现好多问题

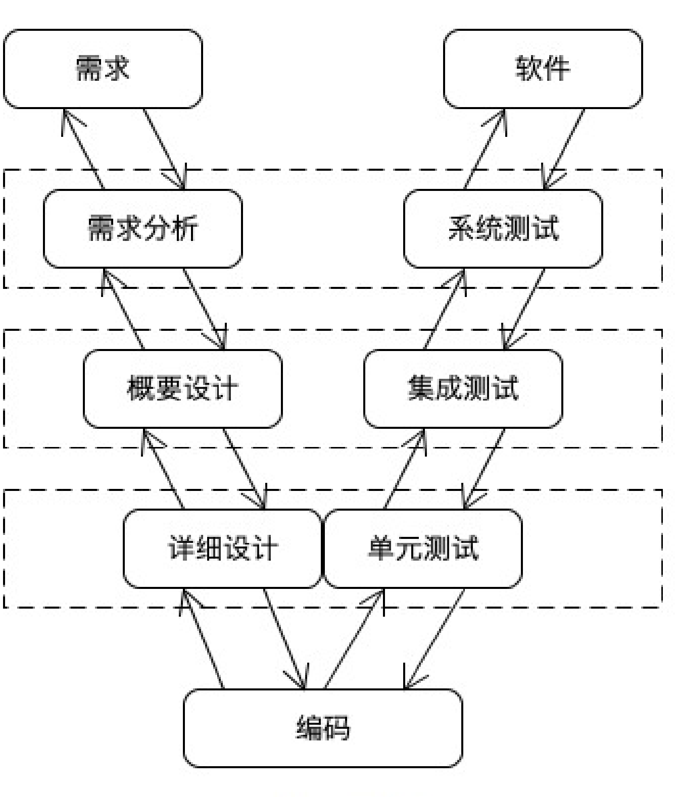
1. 需求文档经常描述不清楚，或者遗漏。
2. 工作人员经常认为编码完了，工作就完了。
3. 各种设计文档工作量很大，内容乏味，但编码完成后基本都对不上了。

参考：https://en.wikipedia.org/wiki/Waterfall_model

V模型

后来V模型出现了。V模型是瀑布模型的变种，其实我也把它归类为瀑布模型。

不确定V模型都做了哪些改变，似乎只是这样变了一个姿势，该做的事情一样不差，但是通过这个姿势，凸显了测试和开发环节的对应关系。表达了——都是软件开发环节的一部分，**测试一样很重要。**



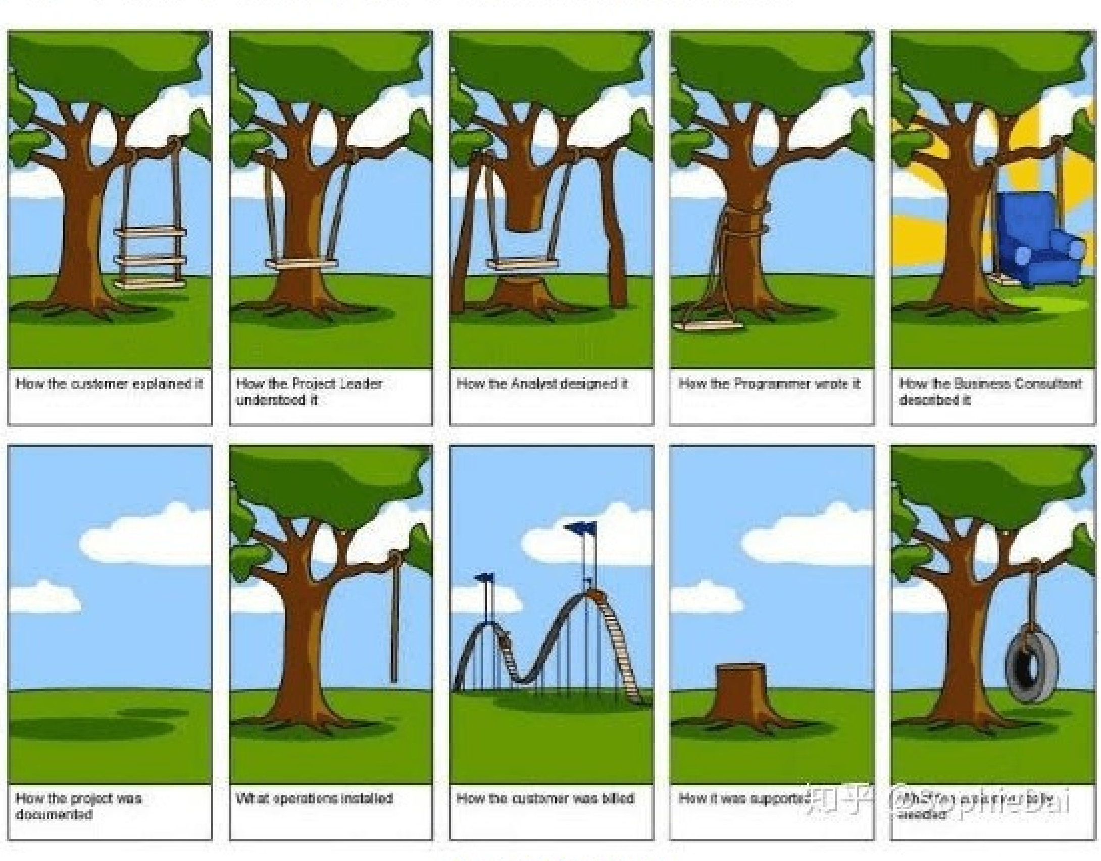
(V model)

我不确定专门的测试部和QA是否是在这个过程中出现的，但可以看出，V模型试图解决瀑布模型中的问题2。

参考：https://en.wikipedia.org/wiki/V-Model

原形模型

在瀑布模型中第一个环节就是“需求分析”，这个环节的产物决定着最终产品的走向，需求的重要不言而喻。有一个出镜率非常高的图，非常准确的描述了错误需求的结果。



(demand-error)

原形要解决的问题就是需求不准，避免需求经过长时间的开发，浪费了大量的金钱和人力，得到的软件还不是用户所期望的。

原形模型采用的方式是：开发团队在分析需求的时候，尽快开发出一个用户看得到的原形，**让用户尽早感受到效果。**其实原形模型更多的是一种沟通方式，只是有人不丢掉原形，在原形的基础上继续开发，才被定位为原形模型。不过原形的开发过程时间紧，任务重，结果非常粗糙，重用的成本一般很高，建议还是丢掉。

在制作原形的时候，有时会做得很逼真，用户可以像真的系统一样操作，只是后台的逻辑都是假的；有时会做的很简洁，只是一些图片。据说iPad的开发过程使用了原形，那个时候的原形仅仅是一个木板，上面画了几个按钮，想要做什么就在上面假装点点，想象着它完成了想要的操作。只要有好的沟通效果，形式不重要。

经过这一轮操作，用户基本就确定了自己想要什么了，完美的解决了瀑布模型中问题1。

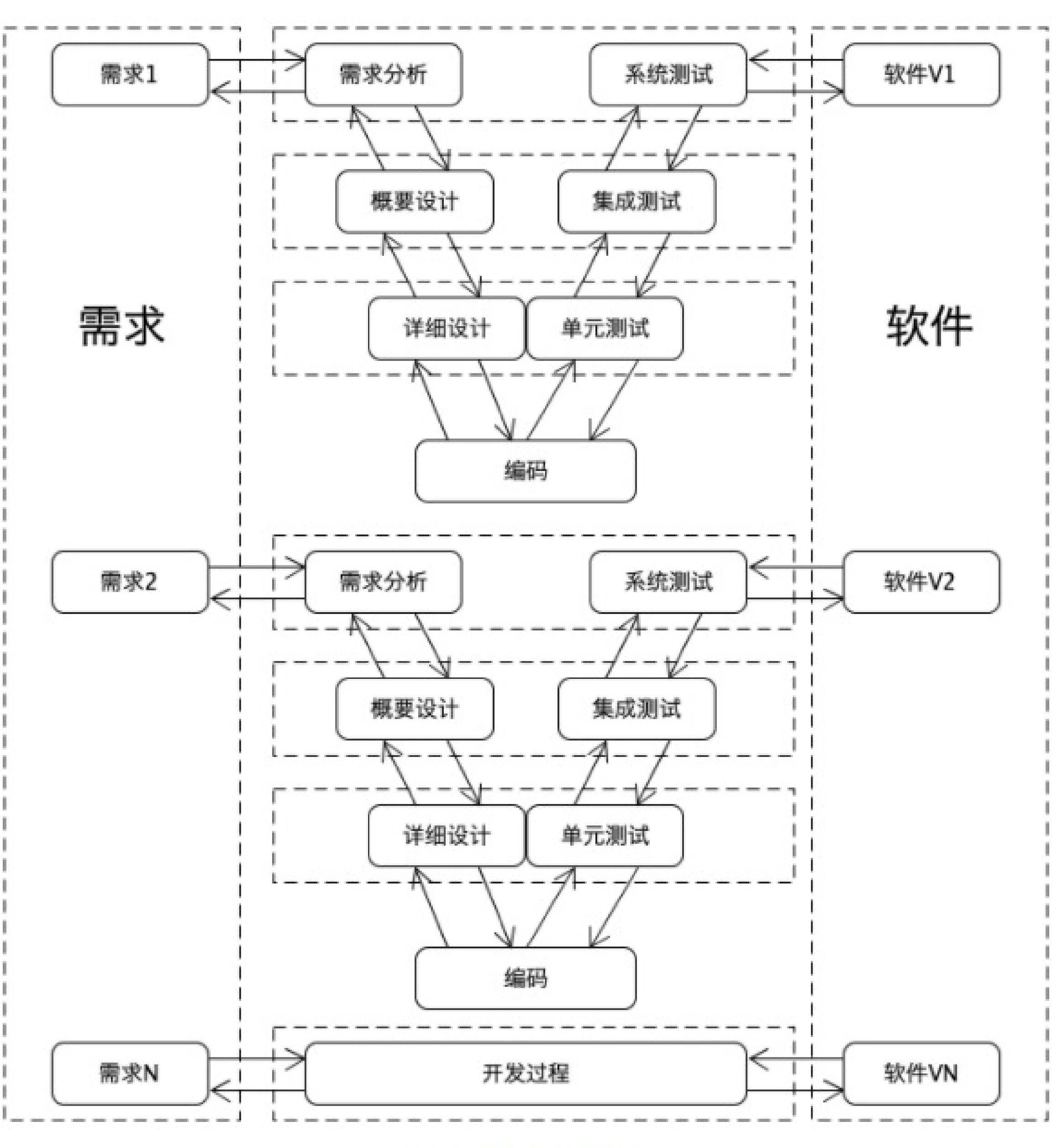
参考：https://en.wikipedia.org/wiki/Software_prototyping#Throwaway_prototyping

迭代模型

迭代模型的思路是分解需求。看似简单的分解操作，却得到了三个好处：

1. 当需求变小后，每个需求的开发过程就会变简单，每个阶段的工作也都可控了。每个迭代的需求都像瀑布模型一样有分析、设计、开发、测试，但是因为需求小，对文档的依赖减弱很多。
2. 开发人员可以将前一个迭代学到的东西用在下一个迭代，开发越来越顺畅。
3. 为开发不确定需求提供了可能。虽然整个需求没有完全想清楚，但是想清楚的部分可以先开发。

效果如图：



(Iterative model)

这个图上每个迭代我都画成了V模型，实际并不一定是V模型，任何适宜模型都可以。

迭代模型将大的需求变成小的需求，让问题的复杂度降低了很多，瀑布模型中各种设计文档的作用开始减弱了，缓解了瀑布模型中问题3。

参考：https://en.wikipedia.org/wiki/Iterative_and_incremental_development

敏捷

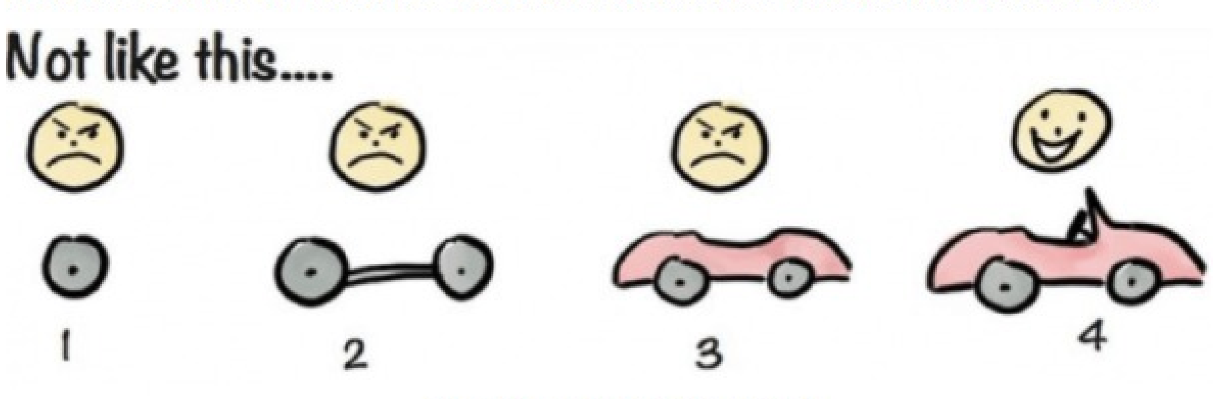
敏捷核心关键词：**快速交付，持续重构；演进的需求和方案；自组织且跨功能的团队。**

快速交付，持续重构

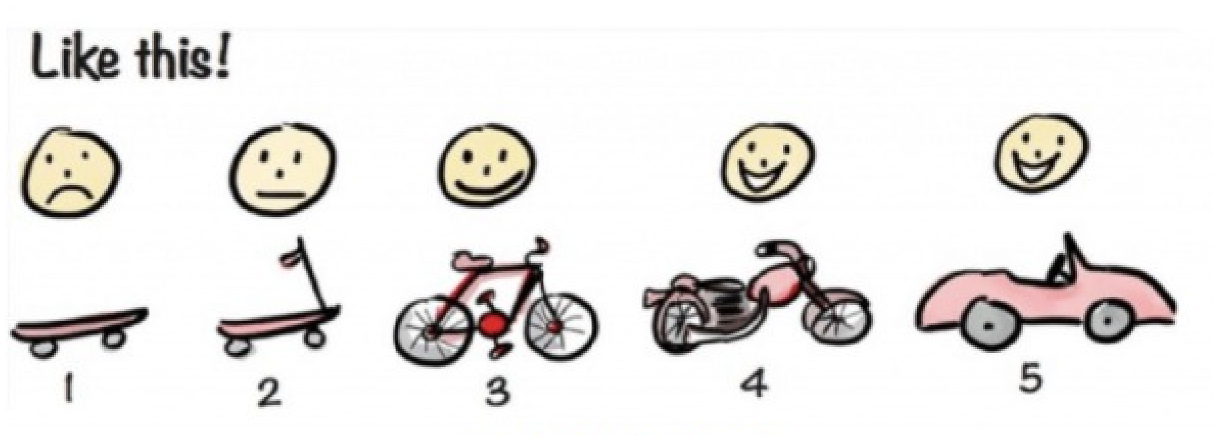
- 通过TDD过程，在开发需求之前就细化了需求的细节，对详细设计的结果有提升。
- TDD产出的测试用例，提升编码质量，避免反馈周期长。如果没有测试用例，一个Story很容易被当皮球踢。
- 项目中遗留的测试用例，能避免软件成为打地鼠软件。当增加一个A功能，发现B功能有问题，修复了B功能发现C功能故障了，修复了C功能，发现A功能又故障了。
- 通过Pipeline，让构建、测试更频繁，因为工具化了，成本更低，出错更少。
- 通过Pair、CodeReview，让知识流动起来。

演进的需求和方案

用户的需求一般都只是一个大方向，具体的价值和实现方式都是不确定的，用户也在不断探索他们的业务。敏捷接纳了原型法的需求分析方法，还提出了Inception来分析更有价值的需求，通过MVP圈定最小开发范围，快速验证方案，这就是演进的需求和方案。提到演进我们经常看到这个图：



(traditional demand analysis)



(agile demand analysis)

这个图确实很形象的表达了敏捷和传统开发方式比较，增加了演进的过程，但是大家可能没有注意到需求是什么。

一般出现这个图时的需求是：一个舒适快捷的载人工具。

因为用户不确定最终产品的样子，用户只有一个对产品的期望。开发也不确定各个材料具有什么性能，所以用演进的方法来探索答案。看起来浪费了很多金钱和时间在和汽车不太相关的东西上，这个其实是寻找答案的成本。如果不演进，可能需要5个汽车的成本才能搞定，而演进只花费了两个滑板、一个自行车、一个摩托、一个汽车的成本，所以算下来还是很划算的。**这里不确定的需求是采用演进的原因。**

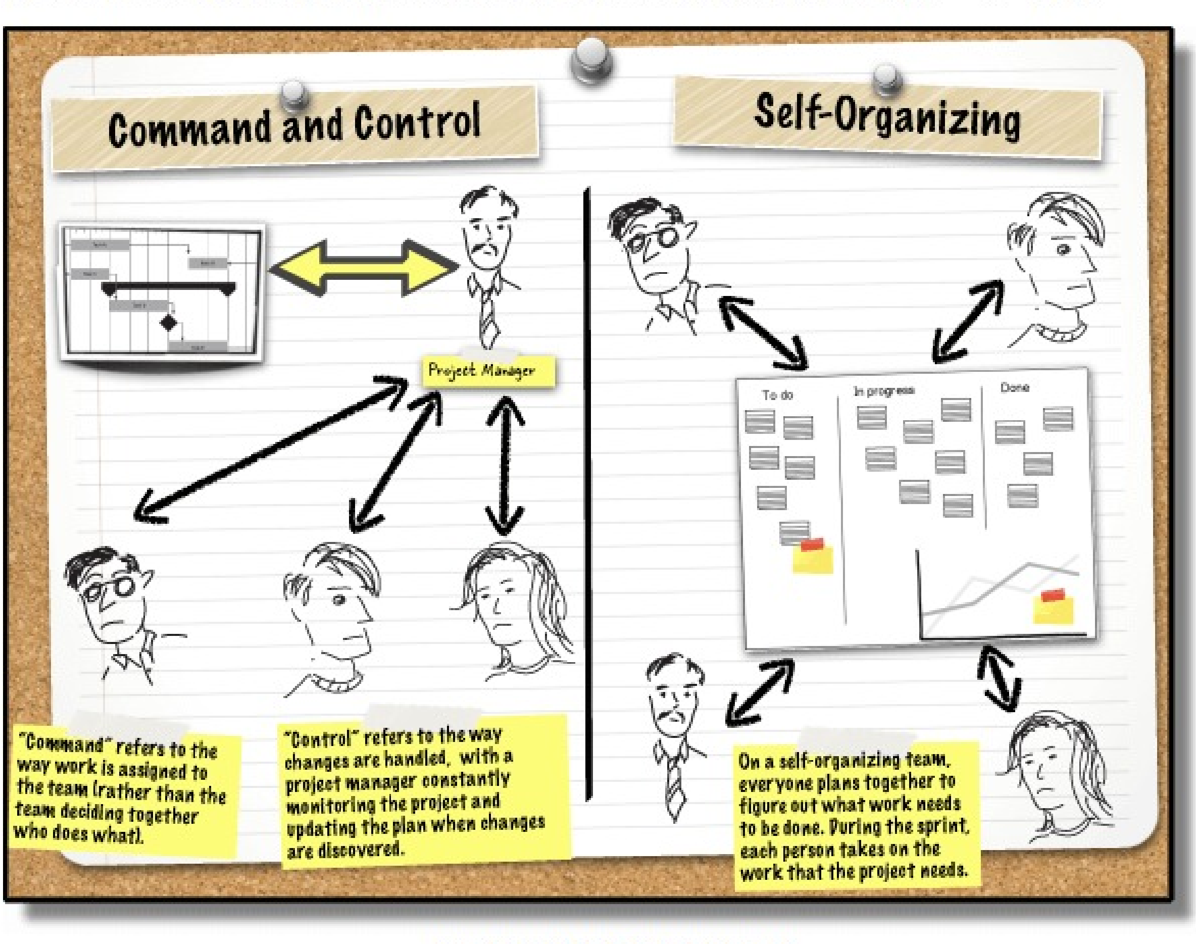
如果需求是确定的——有四个轮子，有发动机，有顶棚和方向盘的汽车。我们应该可以直接使用第一个流程，第一个流程能节省更多的钱和时间。敏捷并没有说所有的需求都要演进，确定的需求还是设计的好，不确定的需求就需要演进。

但是作为一个需求，一般不会所有的东西都是确定的，比如这个需求里面仅仅要求有四个轮子，轮子是否需要有轮胎呢？在轮子的开发上还是需要敏捷，同理其他模块也一样。

最终的开发过程应该是，由总工程师设计车架（汽车的架构），将需求中的四个轮子、发动机、顶棚和方向盘连接起来，之后多个项目组并行开发其他模块，各个项目组都可以使用敏捷开发各自不确定的模块。大家是否找到了软件的影子，其实我们都是这么干的。比如一个全新的系统，我们会规划一个用户管理服务、基本数据服务、XX业务服务……

自组织且跨功能的团队

将自组织(Self-Organizing)和传统的命令控制（Command and Control）比较一下，如图：



自组织与传统命令控制的比较

Command and Control组织方式下，只有Project Manager关心进度，其他人员专职负责各自分配的任务。

一个复杂的组织就像一个复杂的机器；组织是各个岗位的人组成的，机器是各个位置的螺丝钉组成的；一个优秀的机器，依赖于每个螺丝钉都非常牢靠，一个优秀的组织也依赖于每个岗位的人能非常专业。

一万小时定律说：1万小时的锤炼是任何人从平凡变成世界级大师的必要条件。每个人的精力都是有限的，为了能在某个领域专业起来，必定需要在这个领域长时间专职投入。

所以Command and Control还是挺科学的，它将复杂的组织化繁为简，让每个岗位都专注得在自己的位置上尽职的发挥着光和热，成为优秀的罗螺丝钉，再用优秀的螺丝钉做优秀的机器。

可是，人终归不是一个螺丝钉，他有血有肉，有想法，有个性；工作也不是铁板上打出来的那个孔，一成不变。所以现实中工作和人的匹配总不是那么合适。

在Self-Organizing组织方式下，每个人都致力于项目的目标，团队成员互相尊重，每个人都专注于工作，开放，团队成员有勇气站出来参与该项目。这里最关键的就是**每个人都致力于项目的目标。**如果只有Project Manager直接关注目标，其他人很难知道自己做的内容和目标之间有什么关系，就不能对这个目标做太多的努力。类似一个开发人员开发了一个定时任务需求，测试人员需要等待若干小时才能测试一次。如果开发人员提供一个手工触发的接口，测试人员的工作效率就会提升很多。所以看清楚全局的目标和问题能很大的提升生产力，敏捷的每日站会就在为这个而努力。

自组织内容比较多，有兴趣的可以阅读一下参考链接。总的来讲在自组织的团队中，每个人心中都有一个信念：为了部落。

参考：

https://en.wikipedia.org/wiki/Agile_software_development

https://www.infoq.cn/article/what-are-self-organising-teams

https://cloud.tencent.com/developer/article/1370521

总结

每一个模型都是在已有模型的基础上不断演进的，后来的模型都在继承已有模型优势的基础上解决已有模型的问题。

敏捷继承了以前模式的各种优点，它没有否定其他模式，它是在改进其他模式。敏捷也被质疑，认为对人的要求太高，实施难度大，但确实解决了不少问题。从这个角度看瀑布就像是一把石斧，敏捷是一把钢斧，他们都是砍柴的，但是钢斧在所有斧子中从多方面做了改进。

《深入核心的敏捷开发》第10章结语——剑道中有这样一个心决：守、破、离。

书里讲的是学习新东西的过程，我想我们在工作过程中，除了守（学习和使用敏捷），还要破（发现使用过程中的各种问题，寻求解决方法），总有一天会离（创建新的模式）。

所以当前敏捷是王道，但是未来一定会出现更好的模型。

其他参考
https://en.wikipedia.org/wiki/Systems_development_life_cycle

https://www.ukessays.com/essays/information-technology/the-history-of-the-waterfall-model-information-technology-essay.php

https://en.wikipedia.org/wiki/Spiral_model