V4版本存在的一些问题：

1. 可追溯性不够好：TestItems中， Current具体对应哪个Recipe，是靠名字来判断的，关联性不够强。（比如TestItems中，Current为D03，对应的Recipe是Oppo\_BLP663\_D03\_CD。）
2. TestItems不可复用：一个Test Item只能属于一个Test Plan，不够合理
3. TestPlans不可复用：一个Test Plan只能执行一次，再次执行需重新建立
4. TestPlan被打散了放到Scheduler里，所以从Scheduler里看某一个TestPlan对应的Raw data比较难找

改进：

1. Tester, Chamber和Batteries都属于Assets，它们的数量是有限的，一旦被使用，就会使空闲资源变少。而一旦当前任务需要的资源不足，那么后面的任务就都处于waiting状态。所以每个Asset都要有一个Status信息和使用记录。
2. 将Test Plan和Test Item改名为Program和Sub Program，因为本质上它们是需要被执行的某种流程。将Recipe划分为Chamber Recipe和Tester Recipe。
   1. What? Program, SubProgram, Recipe (ChamberRecipe, TesterRecipe)本质上都是预先设定好的流程，可以反复被执行。
   2. Why Program? Program是对程序总的规划，比如建立RC Table，是要在n个温度点和m个电流点下实验m\*n（实际可能不止）次。那么这m\*n次实验应该被归纳到一个Program内，以便反复执行。比如我们要对某电池建立RC table，只要执行一个名为”RC Table Generation”的Program就好，而不是要求执行m\*n次具体的Experiment。
   3. Why Sub Program? Sub Program是Program中的某一个电流点和温度点下的一个实验。之所以需要Sub Program，而不是直接使用Recipe组成Program，可通过以下示例说明：-5度500mA实验，如果-5度不在电池可充电温度范围内，那么必须先在可充电温度内充到满，再在-5度下用500mA放电。以现有的设备来说，这必须拆分成两次实验才能完成。也就是得用两个不同的Recipe。假设这两个Recipe是A和B，如果”RC Table Generation” 这个Program下直接包含若干个Recipe，那么如何保证A和B能在执行的时候被理解为一个连续的实验就是个麻烦的问题。因此，一个Program由若干个Sub Program组成，一个Sub Program由若干个Recipe组成。
   4. Why Chamber/Tester Recipe? Recipe即规定了电流又规定了温度，但目前的Tester只能控制电流，Chamber只能控制温度。为了更好的可追溯性，需要把Recipe拆分。TesterRecipe对应到Tester中预先保存的Recipe，ChamberRecipe对应Chamber中的Recipe（因为过于简单所以不需要在Chamber中提前设定）
   5. Recipe, Sub Program, Program都有一个预估时间Estimate Duration，
      1. Recipe Estimate Duration=Valid Duration sum/execution time
      2. Sub Program Estimate Duration = sum(RED)
      3. Program Estimation Duration = sum(SPED)
3. What and Why Request? 当系统里设定好了Program后，Requester需要做的就是向BC Lab申请执行某一个Program。一旦某一个Program被申请执行，那么我们便要开始跟踪和记录执行的情况。显然这些状态是不应该跟Program绑在一起的，设想如果两个Requester同时申请执行同一个Program，而现有资源又允许同时执行，那么绑在这个Program上的状态怎么能表示两组信息呢？或者一个Program执行完后，其状态被更新。到下次再次执行此Program时，原来的状态不是要被覆盖了？

所以我们需要引入Request这个概念（也可以理解为Order订单），用来记录和跟踪被申请的Program的执行状态。

How does Request work?

* 1. 建立Request时，可设置Program ID及其Priority。此Program下的所有Sub Program和Recipe即继承此Priority。此后用户可按需调整。
  2. Request指向Program，Program指向Sub Program，Sub Program指向Recipe，Recipe指向Result。Result即此Recipe实际执行的状况，包括Status, Battery ID, Tester ID, Chamber ID, Start Time, End Time, Raw Data等等。
  3. Request中的Recipe会被添加到Scheduler中去。
  4. Requester可将已完成的recipe状态由Complete改为Invalid。此时会重新为此recipe生成一个Result，并将此Recipe添加到scheduler中去

1. Why Scheduler? Assets是有限的，而Requests可能是并发的，所以我们要用Scheduler管理和记录Assets的使用情况，合理规划可用资源。

How does it work?

* 1. Requests被打散成Recipe添加到Scheduler中来，此时命名为Task。
  2. 根据Priority，对Task排序
  3. 同样的Priority，按Chamber Recipe分类
  4. 用户可在Scheduler中手动调整顺序（Optional）
  5. 属于同一个Sub Program的Recipe不可被分开或改变顺序。见2.c
  6. 分配资源给最优先的Task，从而使其得到执行
  7. 显示被占用的资源和闲置的资源
  8. 显示所有Task的状态，以及预估时间（见2.e）
  9. 完成的Task会从Scheduler中移除