

如何在低成本人员的情况下开发大规模项目

1. 简介

1.1 目的

本文主要就是结合如何开发大规模项目、如何使用低成本人员这两个问题，结合自己实践中的经验，提出一种操作性强的在使用低成本人员的情况下开发大规模项目的软件过程（简称为 LSLC），希望能够给正在或者即将进行此类项目的公司提供过程上的解决方案，降低开发大规模项目、使用低成本人员带来的风险。

1.2 范围

LSLC 主要针对具有以下这些特征的软件项目：

(1) 项目组成

项目组中只有少数人员（项目经理、系统分析员和架构设计师）有比较丰富的项目经验，比例小于 20%；项目组中的其他人员对项目当前的业务知识、技术知识不熟悉，缺乏相关经验，比例大于 80%。LSLC 主要针对的是大量使用低成本人员的项目。

(2) 项目领域

相当于项目整体而言，其所进入的是陌生的业务领域或者技术领域。这是因为如果项目中低成本人员的很多，那么从整个项目所表现出来的能力来看，进入任何一个业务领域或者技术领域，都是一个陌生的领域，都是不具备经验的领域。相对业务领域和技术领域而言，LSLC 侧重于业务领域。

(3) 项目规模

LSLC 对大规模和中小规模的项目都是适用的，但是，考虑到在大规模项目中使用低成本人员的风险要大得多，所以 LSLC 更多的考虑到如何适应大规模的项目。

(4) 客户关系

任何项目的成败，关键因素之一就是客户关系，LSLC 也不例外，并且 LSLC 更强调客户关系，LSLC 对客户关系有以下要求：项目组和客户的关系比较融洽，相互信任；项目的客户比较成熟，理智。

1.3 定义、首字母缩写词和缩略语

LSLC : Large Scale Low Cost Software Process 的前 4 个首字母缩写

1.4 概述

本文主要分为 2 个部分，第 2 部分是 LSLC 的介绍，第 3 部分是 LSLC 的分析总结。在第 2 部分中，主要介绍了 LSLC 的组队模型、迭代模型、开发流程和最佳实践；在第 3 部分中，主要是介绍使用 LSLC 的注意的问题以及使用低成本人员开发大规模项目本身固有的一些问题。

2. LSLC 的介绍

2.1 LSLC 整体说明

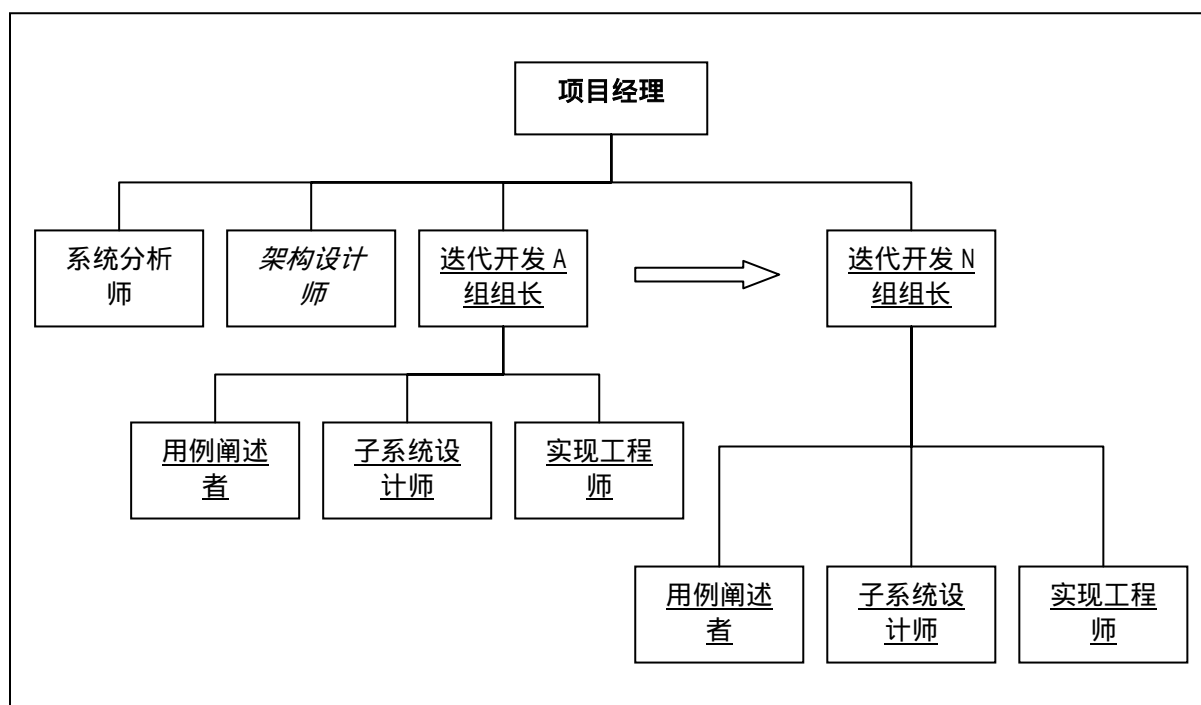
LSLC 主要包括 4 个部分：组队模型、迭代模型、开发流程和最佳实践。

LSLC 是 RUP 经过裁减得到的，裁减的依据是：

- (1) 项目属于大规模的软件项目；
- (2) 项目组进入的是陌生的业务领域；
- (3) 项目组中低成本人员大于 80%；
- (4) 项目组拥有信任的客户关系。

2.2 LSLC 组队模型

LSLC 的组队模型，采用的是 3 + N 的方式，即项目经理、系统分析员、架构设计师加上多个迭代开发组，如下图：



图中是主要的角色，其它的一些角色，比如集成工程师、测试工程师、配置管理员等都没有包括在内。

2.2.1 LSLC 组队模型的描述

以下是各个角色的简单描述：

角色	职责	是否是低成本人员
项目经理	整个项目	否
系统分析师	软件需求的完整性和一致性	否
架构设计师	软件架构的完整性和一致性	否
迭代开发组长	本次迭代	是
用例阐述者	编写每个用例的用例描述	是
子系统设计师	详细设计各个子系统，设计类	是
实现工程师	实现子系统，实现类	是

2.2.2 LSLC 组队模型的优点

LSLC 组队模型主要考虑的问题是，如何配备人员，充分挖掘有经验人员的能力，让他们来承担项目中重要、关键的工作，解决项目中的问题和风险，以使其它成员能够顺畅、无障碍的进行剩余的工作；同

时，在有经验的人承担项目中重要、关键的工作时，如何尽可能的减轻他们的负担。针对这些问题，LSLC 组队模型的解决方案如下：

- (1) 采取有经验的人员在项目组中担任项目经理、系统分析师和架构设计师的角色，其它低成本的人员担任各个迭代的所有角色这种策略，可以保证系统分析员、架构设计师来统一的进行需求分析、架构设计工作，确保项目方向的正确性，确保需求、架构的完整性和一致性。
- (2) 在需求中采取系统分析员和用例阐述者角色分开的策略，系统分析员只是捕获出用例，用例阐述者负责编写每个用例的用例描述。这样做，一是可以减轻系统分析员的工作强度，因为需求中很大一部分的文档工作，即各个用例的用例描述是由用例阐述者来写的；二是可以更有效的保证用例之间的完整性和一致性；三是可以有效的控制需求的质量，因为系统分析员负担减轻后，可以有更多的时间来指导、检查用例阐述者编写的用例描述。

2.2.3 LSLC 组队模型的使用

LSLC 的组队模型具有很好的伸缩性。

对于中小规模的项目，LSLC 组队模型可以按照如下方式进行压缩：

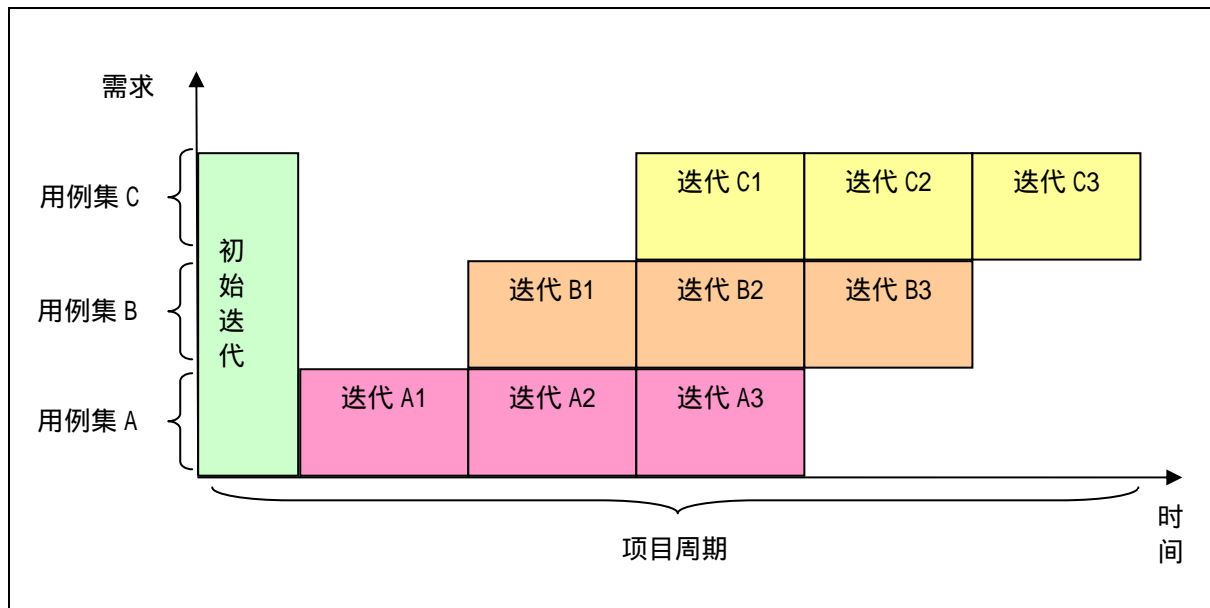
- (1) 项目经理、系统分析师、架构设计师可以由 1 个人或者 2 个人承担；
- (2) N 个迭代开发组可以压缩为 1 个迭代开发组；
- (3) 迭代开发组中，迭代开发组组长、用例阐述者、子系统设计师、实现工程师角色可以由最少 1 人承担。

对于更大规模的项目，LSLC 的组队模型可以按照如下方式进行扩展：

- (1) 系统分析师扩展为一个组，这个组包含多个系统分析师；
- (2) 架构设计师扩展为一个组，这个组包含多个架构设计师；
- (3) 迭代开发组下面增设 2 级迭代开发组；
- (4) 如果还是不够，继续按照以上的方式往下分级。

2.3 LSLC 迭代模型

LSLC 的迭代模型，采用的是 1 + N 的方式，即初始迭代加上多个并行的迭代。如下图：



上图是一个示意性质的，并不代表只能划分 3 个用例集，每个用例集只能进行 3 次迭代。玫瑰红的迭代，即迭代 A1、迭代 A2、迭代 A3，组成迭代 A 系列，同样的有迭代 B 系列、迭代 C 系列。

2.3.1 LSLC 迭代模型的描述

LSLC 的迭代模型，采取的是先整体后部分的迭代策略。即 1 + N 中的 1 是整体，N 是部分。在图中，1 表示初始迭代，N 表示对应于用例集的各个迭代系列：迭代 A 系列、迭代 B 系列、迭代 C 系列。同时，也可以看到，N 所表示的迭代在某些时间点上同时进行的，也就是说，N 所表示的迭代是并行的。

LSLC 的迭代模型中的 N，虽然采取的是并行迭代的方式，但是各个并行迭代间不是同时开始的，而是有先后顺序的。在图中，可以看到迭代 A 系列对应需求用例集 A，迭代 B 系列对应需求用例集 B，迭代 C 系列对应需求用例集 C。迭代 A 系列最先开始，迭代 B 系列比迭代 A 系列晚，迭代 C 系列比迭代 B 系列又晚，这就说明，迭代 A 系列首先定义用例集 A，开发用例集 A，然后迭代 B 系列定义用例集 B，开发用例集 B，然后迭代 C 系列定义用例集 C，开发用例集 C。

LSLC 的迭代模型，对应于各个用例集的迭代系列本身，采用的是递增式的迭代模式。即用例全部定义，用例逐步开发。以对应需求用例集 A 的迭代 A 系列为例，在迭代 A1 中将用例集 A 全部确定下来，在迭代 A1 中开发用例集 A 的一部分用例，在迭代 A2 中开发用例集 A 的另外一部分用例，在迭代 A3 中开发用例集 A 剩余所有的用例。

2.3.2 LSLC 迭代模型的优点

LSLC 迭代模型主要考虑的问题是，如何减少开发大规模项目带来的风险，减少使用低成本人员带来的风险，如何集中有经验的人员并形成优势力量，如何减少项目组进入陌生的业务领域的风险。针对这些问题，LSLC 的主要解决方法如下：

- (1) 采取迭代模型，可以让低成本人员在迭代中进行学习，并且在下次迭代中得到提高。迭代提供了培训人员的容器，也提供了改正错误的机会。迭代提供了快速获得反馈的机制，而正是反馈，才能让项目组知道风险是否已经消除或者问题是否已经解决。迭代能够及早的发现并解决问题，迭代能够让项目不断的纠正自己的方向，保证项目行驶在正确的轨道上。
- (2) 采取先整体后部分的迭代策略，在大项目中尤其重要。很类似于现在经常提的“整体规划，分步实施”的概念。采用这种方式，能首先在大的方向上把握项目，为以后分步开发提供基础。初始迭代定义项目的范围框架，能够保证在以后的迭代中有据可循。

- (3) 采用具有先后顺序的并行迭代，使项目组成员逐步的了解业务，熟悉业务，给项目组学习业务的机会，因为根据项目组成员的实际水平，不可能在项目一开始就能够熟悉全部业务，只能逐渐的学习。
- (4) 采取并行迭代的方式，可以充分有效的平衡资源，快速开发。并行迭代也让低成本人员尽早的参与实际工作。
- (5) 迭代系列中采用的递增式的迭代模式，因为只在各个迭代系列的第 1 次迭代中定义所有用例，所以，能够保证在每个迭代系列中，只有第 1 次迭代是需要投入比较多的系统分析师和架构设计师的资源，而在以后的迭代中就可以投入很少。这样做，可以充分利用系统分析师和架构设计师，让他们在做完迭代系列 A 的第 1 个迭代后，马上可以抽出来进入迭代系列 B 的第 1 个迭代。
- (6) 迭代系列中采用的递增式的迭代模式，可以让习惯于瀑布模型开发的团队更平滑的过渡到迭代模型上。因为递增式的迭代模式，用简单的话讲，就是一次把所有的需求都定义清楚，然后先选择一块需求进行开发，开发完成后，再选择另外一块需求进行开发，这种做法，其实是现在很多使用瀑布模型的团队在开发大规模项目时的做法。

2.3.3 LSLC 迭代模型的使用

LSLC 的迭代模型具有很好的伸缩性。

对于中小型项目，LSLC 的迭代模型可以按照如下方式进行压缩：

- (1) 将用例集 A、用例集 B、用例集 C 合并为一个用例集，相应的将各个迭代系列合并为一个，自然也就不需要采用并行迭代。

对于更大规模的项目，LSLC 的迭代模型可以按照如下方式进行扩展：

- (1) 在每个用例集的迭代系列中，再划分出 1 + N 的迭代。即在各个迭代系列下再分级，下一级包含完整的 1 + N 迭代。

对于有经验人员比较多的项目，LSLC 的迭代模型可以按照如下方式进行改进：

- (1) 同时开始 LSLC 中的各个并行迭代。即同时开始做迭代 A 系列、迭代 B 系列、迭代 C 系列，迭代 A 系列、迭代 B 系列、迭代 C 系列之间不再有先后顺序。

对于时间允许的在业务、技术、人员上风险大的项目，LSLC 的迭代模型可以按照如下方式进行改进：

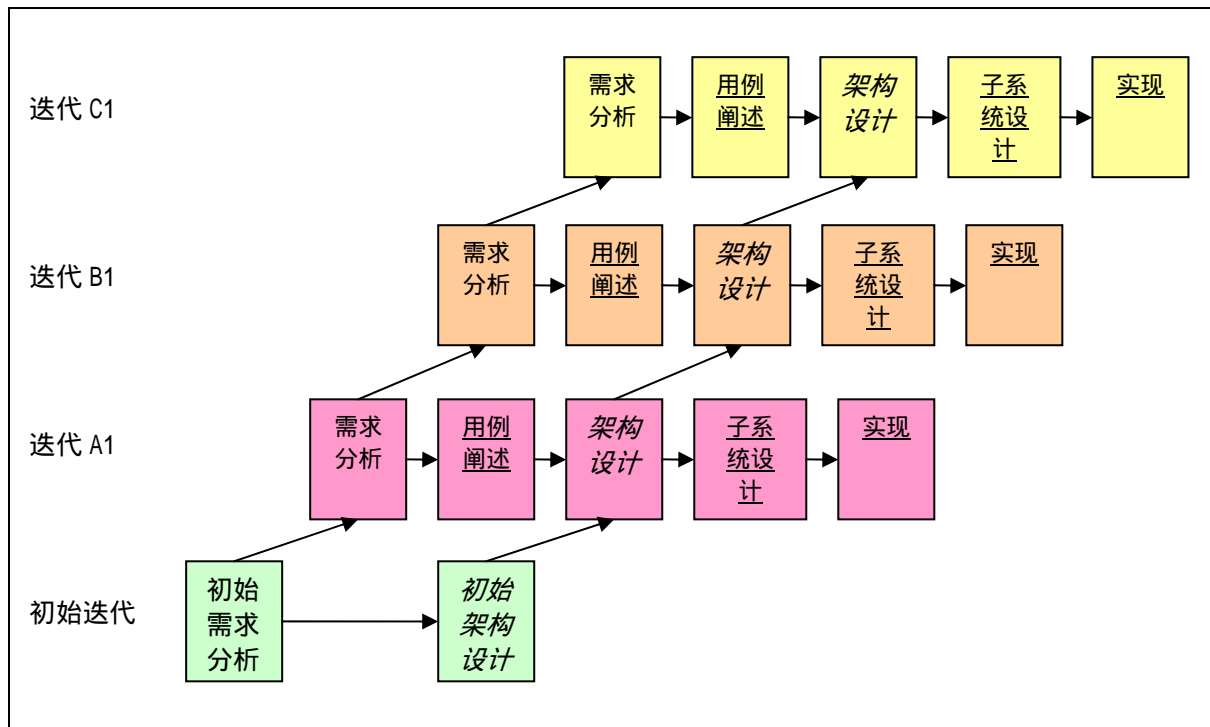
- (1) 串行开始 LSLC 中的各个迭代系列。即迭代 A 系列完成后，再开始迭代 B 系列，迭代 B 系列完成后，再开始迭代 C 系列。

LSLC 采用的并行迭代，要求在划分用例集的时候，各个用例集的关联要尽可能的小。

LSLC 迭代系列中的第 1 个迭代（即迭代 A1、迭代 B1、迭代 C1），如果超出计划时间大于预期值，那就说明项目组能力和项目要求的能力相差较大，此时要么更换人员，要么当机立断，终止项目，避免更大的损失。

2.4 LSLC 开发流程

LSLC 的开发流程，在各个迭代系列之间如下图，只包括初始迭代、迭代 A1、迭代 B1、迭代 C1 的开发流程：



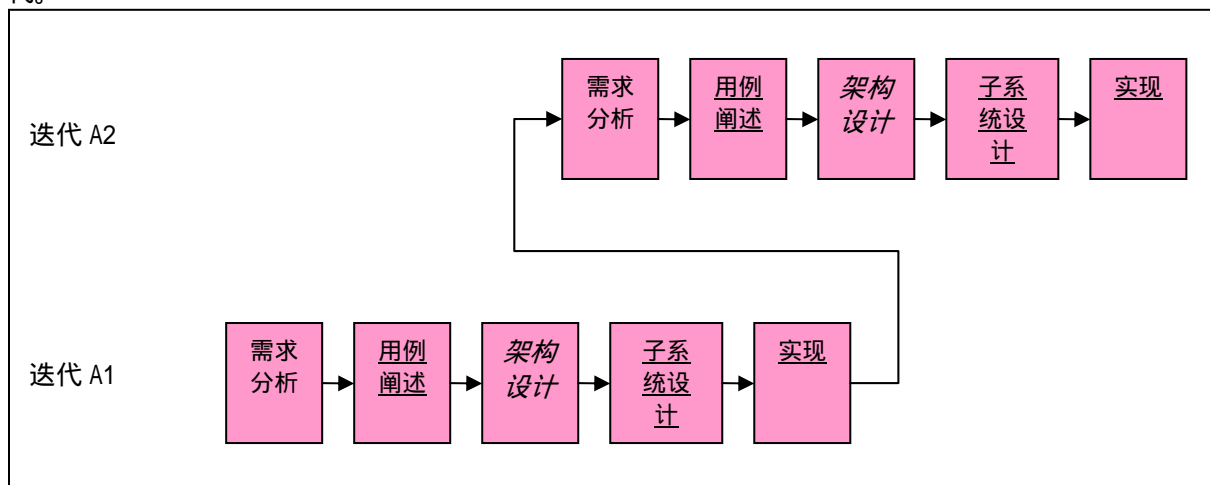
上图中，所有字体为宋体的，均为系统分析师的职责；所有字体为斜体的，均为架构设计师的工作；所有字体加下划线的，均为迭代开发组的工作。这仅仅是一个示意性质的图，主要是为了表示各个角色是如何参与到各个迭代系列中的。

可以看到，系统分析师在完成迭代 A1 的需求分析工作后，就开始参加迭代 B1 的需求分析工作，而迭代 A1 同时进行用例阐述的工作。同样，可以看到架构设计师在完成迭代 A1 的架构设计工作后，就开始参加迭代 B1 的架构设计工作，而迭代 A1 同时进行子系统设计工作。

所有的需求分析工作都是由系统分析师完成，保证项目中关键、重要的工作是由有经验的人来做，同时，也保证需求的完整性和一致性。

所有的架构设计工作都是由架构设计师完成，保证项目中关键、重要的工作是由有经验的人来做，同时，也保证架构的完整性和一致性。

LSLC 的开发流程，在一个迭代系列之间如下图（以迭代 A 系列为例），只包括迭代 A 系列的 A1、A2 迭代。



上图中，所有字体为宋体的，均为系统分析师的职责；所有字体为斜体的，均为架构设计师的工作；所有字体加下划线的，均为迭代开发组的工作。这仅仅是一个示意性质的图，主要是为了表示各个角色是如何参与到一个迭代系列的各个迭代中的。

要特别说明的是迭代 A2，一般来说，其需求分析这个活动只是根据需求变更进行调整，如果没有变更，那么迭代 A2 直接就从用例阐述开始了。但是迭代 A2 的架构设计比较复杂，在迭代 A1 的基础上进行的同时，要考虑到迭代 A2 新的需求，要考虑其它迭代系列的接口。

2.5 LSLC 最佳实践

LSLC 的最佳实践如下：

- (1) 初始迭代可以快速通过；
- (2) 需求分析中划分系统分析员和用例阐述者；
- (3) 在迭代中演进需求和架构；
- (4) 制定详细的迭代进度计划；
- (5) 通过迭代培训人员；
- (6) 在需求分析完成时做好测试数据的准备；
- (7) 开发界面原型；
- (8) 进行充分的风险分析，进行充分的技术预验；
- (9) 制定详细完全的规范；制定详细完全的模板，特别是编程模板和规范；
- (10) 优先开发底层技术框架；
- (11) 强调代码走查；
- (12) 及时通报进度情况；
- (13) 持续集成；
- (14) 尽早让客户试用，获得反馈；

3. LSLC 的分析总结

在使用 LSLC 过程中，有以下问题要注意：

- (1) 团队的整体水平限于项目经理、系统分析员、架构设计师的个人水平；
- (2) 项目经理、系统分析员、架构设计师负担比较重。这个负担不是工程性的工作负担，而是指导、审查等管理性的工作负担；
- (3) 使用并行的迭代，管理复杂，并且各个并行的迭代系列之间协调困难，工作量比较大。

在使用低成本人员的情况下开发大规模项目，有些问题是其本身固有的，使用任何软件过程都不能解决，LSLC 也不例外：

(1) 项目的成功率比较低。

因为在大规模项目中使用低成本人员，不管采用什么软件过程，失败的可能性还是比较大的。所以，LSLC 也不能改变这种规律，它能做的只是针对已经决定在大规模项目中使用低成本人员的项目，提出一种解决方案而已。

(2) 软件过程只能解决部分问题。

在使用低成本人员的项目中，会有很多的问题和风险，解决这些问题和风险，是一个系统工程，要在公司的方方面面进行相应的改进和调整，而 LSLC 本身只能解决部分的风险和问题，期望通过 LSLC 就能解决所有相关的风险和问题，是不可能的。用软件过程来解决人员问题，是比较勉强的，就像一句话所说的那样“A project with poor process and good people is a safer bet than a project with good process and poor people”。

(3) 项目的成本不一定会降低。

因为使用低成本的人员本身并不一定能够降低项目成本，和是否使用 LSLC 无关。在实际中，降低成本的方式有很多种，而使用低成本的人员这种方式来降低成本，是比较具有挑战性，风险性也比较大的一种方式，如果控制不好，会导致成本上升。

总之，LSLC 是针对在使用低成本人员的情况下开发大规模项目的有效的解决方案。它在提供给项目或者组织软件过程的同时，也在软件过程中加入很多防范风险的策略，能够解决大规模项目本身带来的风险，能够解决使用低成本的人员在业务、技术、经验上的缺乏带来的风险，能够集中优势力量，充分带动项目，为摆脱困境提供解决之道。