**工作开展思路汇报**

1. **项目管理**

在标准的敏捷开发中，项目组是“自我管理”的，但前提是：

* 项目组整体经验丰富、职业素养较高，专业、专注、主动，能够实现自我管理、自我引导；
* 项目组经过长期的Team Building，相互之间有相当的配合默契程度；
* 项目组的Leader（Scrum Master）尽职、积极、激情，能作为团队构建者有效激励团队、保护团队不受干扰；

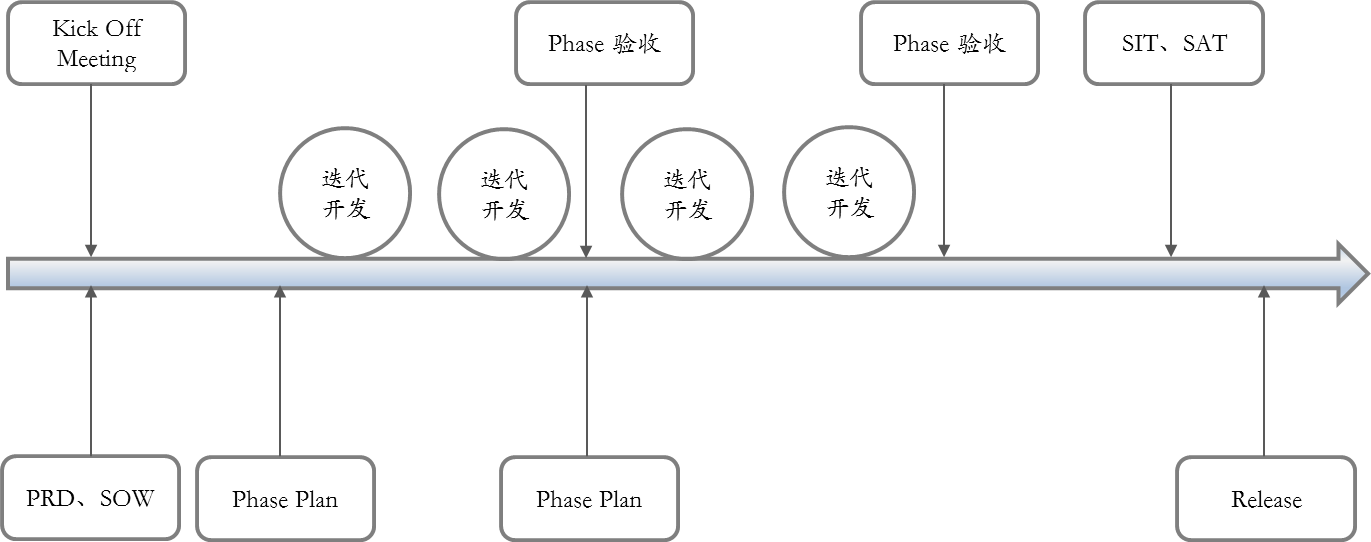
实质上敏捷开发对整个组织结构都有较高的要求，而且因为敏捷开发的特点，导致开发成本也是较高的，以上情况导致在现行情况下，难以实现标准的敏捷开发模式，必须针对组织特点，对相关过程进行裁剪。

裁剪后的流程模型，仍以瀑布模型下的开发团队为基础，设立项目经理等传统岗位，但在日常开发流程中，吸收敏捷开发的部分优点，以相对现阶段提升目标管控、过程管控、质量管控。

* 1. **项目管理目标**

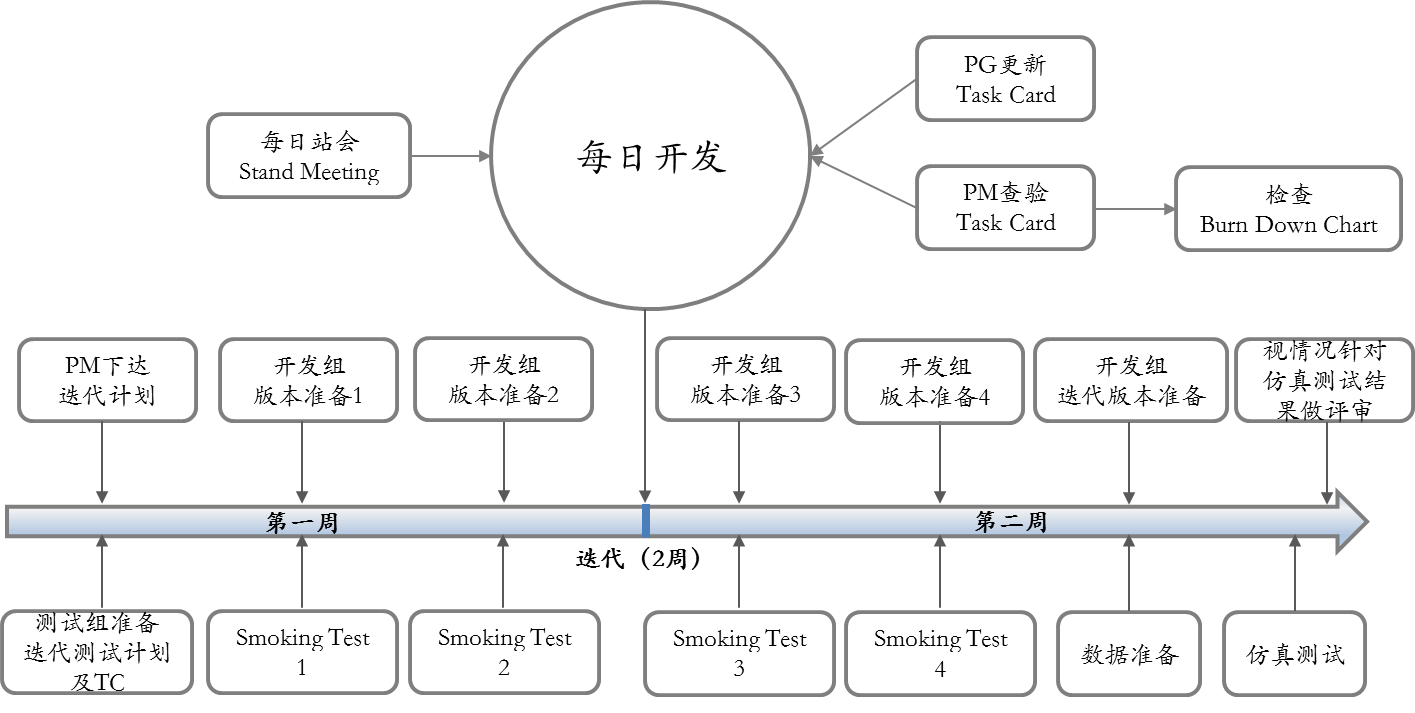
当前项目管理目标主要是

* 理清开发流程；
* 明确项目研发的目标；
* 加强质量控制，及时发现问题；
* 开发过程、各阶段的开发成果、开发过程中的各项变更对各级管理者可见、可控制；
* 关键流程、关键节点加入评审；
  1. **项目流程管理**
     1. **项目周期管理**

****

项目关键活动为：

* Kick Off Meeting ：召集系统研发的管理、商务、产品、设计、研发、测试各方召开的项目启动会，目的在于知会各方项目正式启动，并就相关事项达成一致；
* Phase Plan ：阶段开发计划，本阶段需要实现的项目目标，一个Phase可包含多个Iteration；
* Phase验收：验证Phase产出、对比产出与需求的偏差；
* SIT ：系统集成测试；
* SAT ：现场验收测试，以客户方环境、数据、TC为标准进行的测试；
* Release ：通过SAT后，发布实地部署版本等工作；
  + 1. **迭代周期管理**

****

项目关键活动：

* Sprint Plan ：迭代计划，Sprint Plan是根据Phase Plan分解得到的，以Story的方式呈现的子计划。标准的Scrum流程里面是经过整个Scrum Team讨论得到计划，并由成员自行选择要执行的任务，裁剪后的流程是项目组管理人员：PM、Arch、TL讨论后得到的计划；
* Task Card ：Sprint Plan给团队成员下达的任务，以任务卡片（Task Card）形式下发（通过Scrum管理工具） ；
* Stand Meeting ：PM每天早上10点之前，召开stand meeting，项目组的每个人都简要讲述昨天的工作、出现的问题、今天的计划。每人发言不超过3分钟，所述核心内容需要记录在白板上，如涉及到需要详细讨论的问题，等单独开会再讨论，保持stand meeting的简洁性。会后需要将当天白板内容拍下来发给项目相关人员，或者直接发送会议纪要；
* Smoking Test ：冒烟测试也称为程序健康度测试，主要用来快速检测临时版本是否满足提测要求、是否能满足开发调试要求。测试人员从三个方面检验临时版本：是否满足需求、是否能正常执行功能、是否能正常执行流程。通过Smoking Test的版本才允许正式合并到Master版本中；
* 仿真测试 ：在迭代末尾，从用户角度进行的测试及演示。仿真测试通过使用“干净”的数据库（不能使用开发人员提供的数据库）、真实的初始化数据、真实的用户数据，以用户流程角度而非程序功能角度来测试软件功能。测试人员在程序通过该测试后，需要召集PO、Supervisor、用户代表等高层角色来演示系统，通过各方面不同的视角来检验软件是否满足需求。完成仿真测试的软件版本将被放置在单独的演示服务器中，不受开发、测试的干扰，直到下一个迭代的仿真测试通过才更新版本；
  + 1. **里程碑、评审**

在项目相关流程进行到特定阶段，达成里程碑，产出重要产物时，需要召集PO、Supervisor等高层角色进行评审，以确保项目研发方向的正确性。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **里程碑类型** | **里程碑名称** | **汇报方式** | **负责人** | **汇报成果** |
| 管理 | 项目立项 | 会议 | Supervisor  PO  PM  Arch | 项目立项报告 |
| 设计 | 客户调研 | 会议 | PO  PM  Arch | 调研报告 |
| 管理 | 组建研发团队 | Email | PO  PM  Arch | 人员联系名单 |
| 设计 | 项目需求确认 | Email | PM | 需求规格设计说明书 |
| 设计 | 项目设计确认 | 会议 | PM  Arch | 概要设计说明书 |
| 设计 | 项目开发计划 | 会议 | PO  PM | 项目开发计划 |
| 设计 | 阶段开发开始 | Email1 | PO  PM  Arch | 阶段开发计划 |
| 设计 | 迭代开发开始 | Email | PM  Arch | 迭代开发计划 |
| 测试 | Smoking Test | Email | PM  TL | 测试功能列表、报告 |
| 研发 | 项目迭代验收 | 会议 | PM | 迭代版本功能列表 |
| 测试 | 仿真测试 | 会议 | PO  PM  Arch  TL | 仿真测试报告 |
| 研发 | 项目阶段验收 | 会议 | PO  PM | 阶段版本功能列表 |
| 测试 | 项目验收测试 | 会议 | PO  PM  Arch | 版本缺陷报告 |
| 实施 | 项目版本预备提交 | Email | PM  TL | 实施计划 |
| 实施 | 客户反馈审核 | Email | PM  TL | 需求变更报告 |
| 实施 | 客户方验收 | 会议 | PO  PM  Arch  TL | 项目验收报告 |
| 实施 | 项目结项 | 会议 | Supervisor  PO  PM  Arch  TL | 项目结项报告 |

* 1. **项目管理工具**

各项目组采用统一项目管理工具，该工具经管理层审核批准后全面实施，该工具需具备如下功能：

* Web模式；
* 具备标准的Scrum管理组件：Backlog、Sprint Board、User Story、Task Card、Burn down Chart；
* 具备基本的任务管理功能，包括：Sprint 时间、Sprint Backlog、任务附件、任务状态管理、任务修改内容的历史记录、任务状态变化记录、对该任务的讨论记录；
* 任务数量、任务时间、效能等数值的分析报告；
* 可自定义子系统、模块、任务状态、任务类型、优先级等字段；

项目管理工具主要提供以下功能：

* PM通过项目管理工具制订Sprint Plan、对组员下达Task Card、通过管理工具与组员就Task进行沟通；
* 项目成员通过项目管理工具获取自己的Task，每日工作结束后在项目管理工具中填写在Task上耗费的时间，对需求及Task的疑问、讨论、修改，都需记录在项目管理工具的Task信息中；
* Arch通过项目管理工具验收项目成员提交的Task，并在Review相关成果后，在项目管理工具中确认Task完成；

推荐的项目管理工具有：

* JetBrains的You Track。You Track可以和JetBrains自己的TeamCity（持续集成工具）进行集成，也可与Jenkins（持续集成工具）进行集成；
* 微软的TFS；
* 国内公司的Teambition；

1. **开发组织管理**

开发组织仍以传统的瀑布模型为基准，设立项目经理等岗位（具体原因在第一章项目管理中有阐述）。

以下描述的是基准结构，实际执行中会针对性调整。

* 1. **项目组结构、职责**

以上项目组结构，在执行过程中可能出现一人承担多个角色的情况，如项目经理兼任程序经理、程序经理同时兼任资深开发工程师等。

* + 1. **项目经理（PM）**

职责：

* 团队构建者，推动团队建设；
* 管理项目的计划，包括实施计划、开发计划、人力资源计划、培训计划、成本计划以及奖金计划等
* 管理项目的风险，有效识别项目的风险并制定应对措施
* 管理项目的里程碑，对于外部里程碑，需要有专门的预警机制，在里程碑到来前进行认真的分析，对于不能达成的或者有风险的，及时告知相关人员并进行磋商，原则里程碑必须达到。对于没有达到的，需要有详细的里程碑分析报告并提交商务和研发管理层。
* 推进项目的验收。
* 项目组的需求定义者。项目组对需求实现的确定，由PM负责，对于开发团队，PM是最终的需求确认者。PM觉得无法确定的，可以向产品负责人汇报，组织相关人员讨论。开发人员不能越过PM确认需求；
* 协调项目开发中日常管理事项；
* 召开项目的例会和协调会议。
* 定期组织召开项目的成果演示会议
  + 1. **程序经理（架构师，Arch）**

职责：

* 配合项目经理，验收、确认团队成员提交的成果；
* 代码规范制定者，对模块功能定义、整体框架负责；
* 数据库、程序配置、引用等全局性项目，由架构师统一修改并通知团队；
* 外围接口由架构师统一负责；
* 组织Review代码；
* 负责Daily Build，代表开发团队提交正式版本（包括提交测试版本和发行版本）；
  + 1. **资深开发工程师（SE）**

职责：

* 功能模块底层设计；
* 重难点功能研发；
* Review开发工程师任务、代码；
  + 1. **中级开发工程师、开发工程师（PG）**

职责：

* 按照需求、设计文档进行编码工作；
* 按照程序经理、资深开发工程师要求编写相关文档；
  + 1. **测试经理（TM）**

职责：

* 配合项目经理，组织测试工作；
* TC、BUG相关规范制订；
* 测试流程制订者；
* Smoking Test、仿真测试、SIT、SAT的组织者；
* 测试质量控制；
* 对外版本发布；
  + 1. **资深测试工程师（ST）**

职责：

* 需求到TC的解析；
* BUG认定；
* Smoking Test、仿真测试设计；
* 指导中级测试工作成、测试工程师开展测试工作；
  + 1. **中级测试工程师、测试工程师（TE）**

职责：

* 按照相关需求、测试计划，在ST指导下执行测试任务；
* 按照TM、ST要求，编写相关测试文档；
  1. **架构师小组**

架构师小组是一个虚拟的开发组织，由管理人员、项目经理、程序经理等核心技术管理人员构成。组建架构师小组的目的是统一进行部门级的程序架构设计、研发、管理，避免以下情况出现：

* 避免各项目组自行研发项目框架导致的重复研发、成果无法共享等问题；
* 避免架构设计局限于部分人员进行，考虑不够周到，导致后期出现需要重新研发或产生重大变动的问题 ----后期变动过大而没有考虑到兼容性，已使用旧框架的团队将受到影响；
* 避免架构设计中采用不兼容的组件、接口、设计，导致部署、与其他产品对接时产生问题；

架构师小组的工作范畴包括：

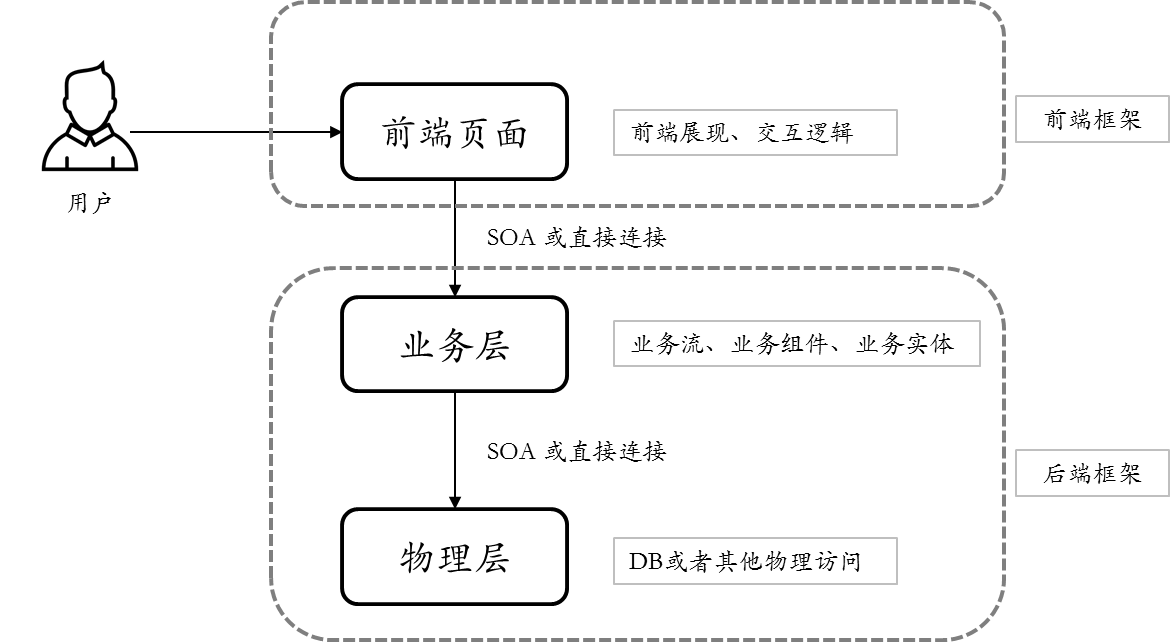
* 制订各项技术规范；
* 部门级架构设计，包括基础前端架构、基础后端架构；
* 指导非标准结构项目的设计；

架构师小组进行部门级架构设计的工作方式为虚拟项目团队，视部门级架构设计为特殊项目，与普通项目一样执行需求分析、设计、开发、测试、部署的全过程。架构师小组因成员非固定成员、所开发项目亦非普通项目，区别于普通项目组之处为：

* 按版本发布架构。下一版本相关目标、特性、改动，需在架构师小组内部讨论通过后先Release文档，以使相关人员做好准备、提供意见；
* 实际开发任务由架构师小组内部成员进行，或交由特定的普通项目组进行。如交由架构师小组内部开发，因架构师小组能实际进行开发的人员不固定，项目开发计划按“最小开发人数”进行；
* 架构设计产物对各项目组公开，非特殊原因，各项目组在项目中需使用该架构设计；

1. **开发框架及工具**
   1. **整体框架**

目前的主流开发架构中，以三层架构较为常见，其主要思想是将程序分成展现、业务、物理访问三个层（Layer）：



当前主流Web研发框架有三种类型：纯服务器端代码框架、HTML + 服务器端代码混合框架、HTML与服务器端代码彻底分离的框架。

另如推广前端和后端彻底分离的技术框架方案，则前端已和.NET、Java没有任何关系，可单独招聘前端开发人员（后端开发人员在此类型框架中相对需求较少）。

* + 1. **纯服务器端代码框架**

典型框架为Asp.Net，特点是将前端HTML封装成服务端控件，开发人员可直接通过C#等服务器端语言控制HTML页面的行为。

优点：技术难度低，开发效率高。开发人员不需要深入了解JavaScript即可进行研发。

缺点：前端展现部分大量运用服务器端控件，美工、UI制作的界面不能直接使用，需要开发人员进行转换。

* + 1. **HTML + 服务器端代码混合框架**

典型框架为Asp.Net MVC，相比Asp.Net，MVC放弃了纯服务器端控件，特点是可在前端HTML页面中直接混入服务器端代码，类似传统的脚本型语言做法。

优点：前端部分使用了部分后端代码（C#或者Java），降低前端编写难度。符合传统型脚本语言开发WEB页面的习惯，不需修改即可直接使用美工、UI制作的界面。

缺点：HTML代码与服务器端代码混合在一起，不易管理。

* + 1. **HTML与服务器端代码彻底分离的框架**

此种框架将前端HTML与后端完全分离，通过Json数据进行交互。前端可用Angular、React等基于JavaScript的框架，后端可使用任何支持Ajax通信的语言开发。

优点：前后端彻底分离，不需修改即可直接使用美工、UI制作的界面。与服务器端语言彻底无关，不受C#、Java的约束。

缺点：开发成本较高，前端交互界面调试缺乏成熟的IDE（集成开发环境）支持。大多数C#、Java的程序员对JavaScript及相关框架的掌握并不熟练，学习成本高、学习曲线陡峭。

对前端要求较高的公司里，大多是将前端工程师与后端工程师分开招聘，需要一定的人力资源上的支持。

* 1. **前端框架**

因互联网最近几年的蓬勃发展，在追求Web前端体验的前提下，前端组件、控件在最近几年风起云涌，前端框架的数量非常之多，国内各大互联网公司也都有自己研发的前端框架。

前端框架的选择主要基于：

* 易用性、文档资料是否完备；
* 基于此框架的UI控件、插件的数量、质量；
* 性能；

目前部分前端框架为实现OO（面向对象）、MVVM（模型视图分离）、双向绑定等高级特性，采用了颠覆性的技术，不再局限于提供DOM操作、动画效果等功能，而是创建了属于自己的一套开发规范、语言特性（如关键字）。导致从基本的JavaScript转向此框架开发时，有相当的学习曲线。同时此类框架大多不能使用IDE（集成开发环境，如VS2015）开发，降低了一部分开发效率，需要谨慎选择。此类框架如：

* AngularJS；
* ReactJS；
* BackboneJS；
  1. **后端组件、控件**

后端组件、控件众多，在普通的业务系统中，使用较多的主要是三类：

* 数据库访问组件。此类组件选择需考虑：适配数据库的数量、ORM方式及易用性、针对特殊的数据库应用的可扩展性、性能、配置复杂程度。目前.NET主流数据库访问组件主要是微软的Entity Framework、NHibernate（由Java的Hibernate移植），另有大量轻量级的ORM组件可供选择，如Dapper等，需视具体应用场景决定；
* IOC（反转控制）组件。此类组件用于配置程序，.NET主流IOC组件是微软的Unity和Spring.net；
* SOA组件及相关组件。此类组件用于提供不同层（Layer）、前后端数据交互的方式。. Net主流SOA组件是微软的WCF、Hessian等；
  1. **开发管理工具**

通用的开发管理工具目前主要分成

* 代码管理工具。如SVN、Git、TFS等；
* 持续集成工具。如Jenkins、TeamCity等；
  1. **测试管理工具**

测试管理工具是提供给测试人员、开发人员进行交互的平台，管理方面包括：

* Test Case（TC）测试用例管理、TC与需求的对应关系；
* BUG管理（BUG详细描述、版本测试、回归测试）；
* 质量分析；

选择测试管理工具要求主要是要求使用简洁、功能完备。主流测试管理工具：

* HP QC；
* Mercury TD；
  1. **应对技术转型**

当前主要问题是.NET开发人员招聘困难，有经验的.NET开发人员招聘更加困难的问题。如考虑转向JAVA，有如下方面需要考虑：

* + 1. **转换时间**

成熟的技术团队建立时间一般在2-3年左右

* 如采用当前技术团队整体转JAVA的方式，技术更新时间在半年至一年左右，部门技术框架的建立在2年左右；
* 如采用引入技术团队的做法，技术融合的过程视情况在1-2年左右；
  + 1. **人员风险**

不论是当前技术团队转型，亦或引入新的技术团队，都面临人员流失风险。人员流失风险视控制情况，估计为10%-30%：

* 新人因没有较多负担，还可以学到新东西，比较无所谓；
* 有较强学习能力、喜欢探索新知识的人员转型较顺利；
* 其他人员因个体情况差异较大，较难预估；
  + 1. **技术风险**

现在部门内部已积累了部分前后端技术经验，转向JAVA后，后端技术必然需要全部更新，前端视选择的JAVA技术不同，展现、通讯两部分可能需要更新。

如采用现有团队转向JAVA，在较长时间内可能会出现如下技术风险：

* 技术不熟练导致的时间风险；
* 技术不熟练，深层次出现问题可能性增大（质量风险）；

如采用引入新团队的方法逐步转向JAVA，可能会出现如下技术风险：

* 开发人员对相关产品、技术不熟悉导致的时间风险；
* 产品之间对接，因各自结构的差异产生的技术、质量问题；
  + 1. **成本收益**

|  |  |  |
| --- | --- | --- |
| **项目** | **优点** | **缺点** |
| Java | * 可利用的开源项目多，技术多样化； * 人力资源较丰富； | * 开发时间、开发成本较高； * 人力成本较高； |
| .Net | * 上手快、开发成本较低； * 人力成本低； | * 可利用的开源技术少，大部分要从Java开源项目进行移植； * 人力资源较缺乏； |

* + 1. **推进措施**

现有技术团队手上.Net项目较多，不宜全面展开，建议先引入一个小团队，做相对独立的项目，以此为核心，逐步学习、转化。

另如推广前端和后端彻底分离的技术框架方案，则前端已和.NET、Java没有任何关系，可单独招聘前端开发人员（后端开发人员在此框架中相对需求较少）。

1. **文档管理、组织经验积累**

目前技术设计文档工作比较薄弱，该部分目的主要是明确工作目标 ---- 虽然按照标准流程走需要的文档过多会导致一定程度的开发效率降低，但在不明确目标的情况下进行开发，研发、测试、管理方面很难有一个可共同参考、执行的标准，可能会导致后期修改增多，相比做前期设计的时间，更加影响项目交付。

在技术设计文档和开发效率之间，可通过削减文档数量取得一定平衡，保留最低限度的文档记录。

文档的作用：

* 明确研发目标、记录研发过程；
* 技术标准、产品需求、经验可追溯、可验证
* 方便技术标准、产品需求、经验的传播，让产品各项细节不再局限在某个具体人员的身上，而成为组织的沉淀；
* 降低技术、需求传播的成本；
* 降低人员流失带来的技术风险；

目前建议是做最基础的项目原型、需求分析、数据库设计，其他设计文档如用例分析、详细设计、概要设计可暂时不进行。

* 1. **关键文档管理**

|  |  |  |  |
| --- | --- | --- | --- |
| **分类** | **文档类型** | **文档名称** | **推行阶段** |
| 技术标准 | 项目管理 | 项目管理规范 | 现阶段 |
| 开发文档 | 前端技术框架标准 | 现阶段 |
| 开发文档 | 后端技术框架标准 | 现阶段 |
| 开发文档 | 编码规范 |  |
| 测试文档 | 测试管理规范 |  |
| 项目过程 | 原始需求 | 用户调研记录 | 现阶段 |
| 设计文档 | 产品需求文档 | 现阶段 |
| 设计文档 | 产品原型 | 现阶段 |
| 设计文档 | 用例分析文档 |  |
| 设计文档 | 概要设计说明书 | 现阶段 |
| 设计文档 | 详细设计说明书 |  |
| 项目管理 | 迭代开发计划 | 现阶段 |
| 项目管理 | 迭代开发报告 |  |
| 测试文档 | 冒烟测试报告 | 现阶段 |
| 测试文档 | 仿真测试报告 | 现阶段 |
| 交付与维护 | 实施文档 | 项目交付报告 |  |
| 实施文档 | 项目验收报告 | 现阶段 |
| 项目管理 | 项目结项报告 |  |
| 项目管理 | 需求变更通知单 |  |
| 项目管理 | 需求变更报告 | 现阶段 |

* 1. **客户需求、现场需求、外部需求管理**

在实际项目实施中，客户会提出对现有产品的改动需求，实施人员在实施过程中，也会发现产品需要改动的地方，这两部分外部需求都需要反馈给项目开发人员。

这两部分需求，可能是客户的个性化需求，也可能是对产品改进、革新有重大协助作用的需求，对其做系统的记录有助于产品后期发展。

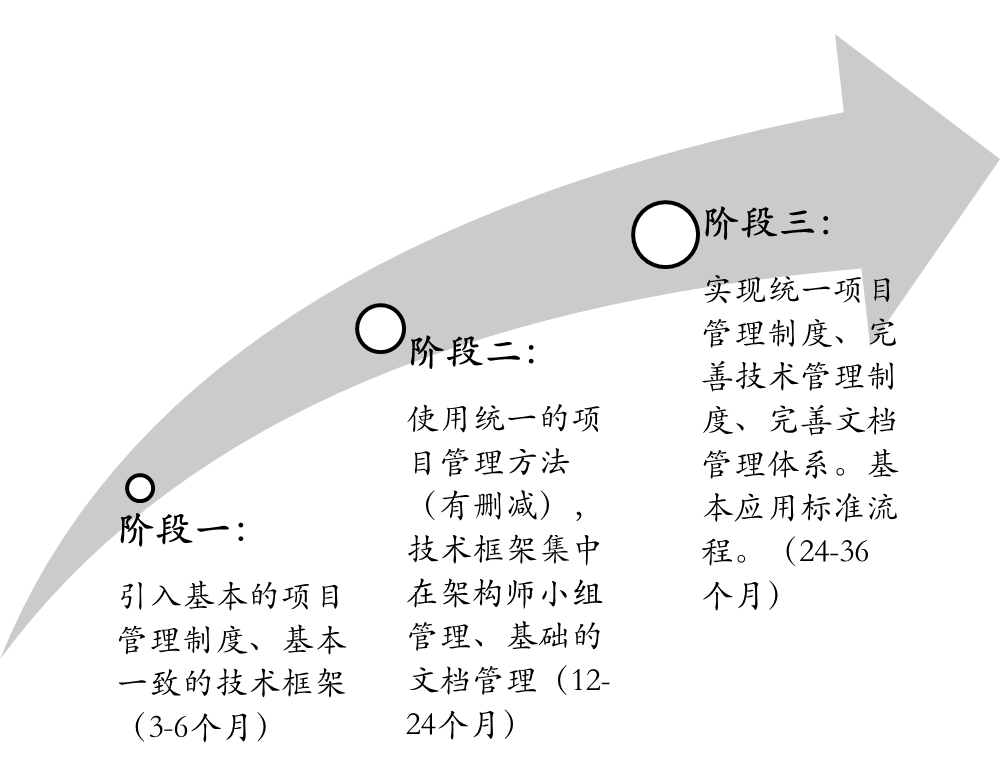
在现形式的开发管理过程中，秉承面向实际需求的态度，用户或其代理也会加入到项目开发流程中，其提出的User Story用户故事会作为Backlog来进行管理，直接进入到项目管理工具中，项目组、PO、Supervisor皆可参与到对用户故事的讨论、分析、需求分解中。

1. **推动过程**

以目前情况，宜分阶段进行流程、技术整合，相比一步到位的做法，成本和风险相对较低。

目前阶段推动内容的中心是：

* 研发流程的规范化；
* 研发基础框架的统一；
* 软件质量的控制；
* 团队技能的整合和提升；
* 相关成果、经验的保存、追溯和传播；
  1. **阶段时间、目标**

****

* 1. **需要得到的支持**
     1. **制度性支持**
* 流程的制订、执行是自顶向下；
* 虽然流程调整过程是柔性的，但具体执行中需要有一定强制性；
* 流程的稳固需要得到制度的支持；
  + 1. **沟通支持**
* 方针政策的确立需要来自管理层的声音；
* 相关流程的确立、调整、宣导需要管理层参与；
  + 1. **管理成本支持**
* 流程的规范，前提是团队建设达到一定程度。相关团队建设，需要投入管理成本；
* 前期的调整，必然带来一定时间成本的损耗，这部分成本需要得到管理层的理解；