

目录

引言.....	2
一、上市保障日常工作.....	2
二、上市保障痛点.....	2
三、解决方案.....	2
四、工具使用示例.....	3
1.流程梳理.....	3
2.事务跟踪在 TaskSteper 系统中的实现(管理员部分)	3
2.1)定义字段 Model.....	3
2.2)定义字段 Form.....	4
2.3)将上述两个步骤定义的 Model 和 Form 添加到 FormAndModelDict 中。	4
3.用户使用 TaskSteper 跟踪问题处理（用户部分）	6
五、进一步优化.....	9

引言

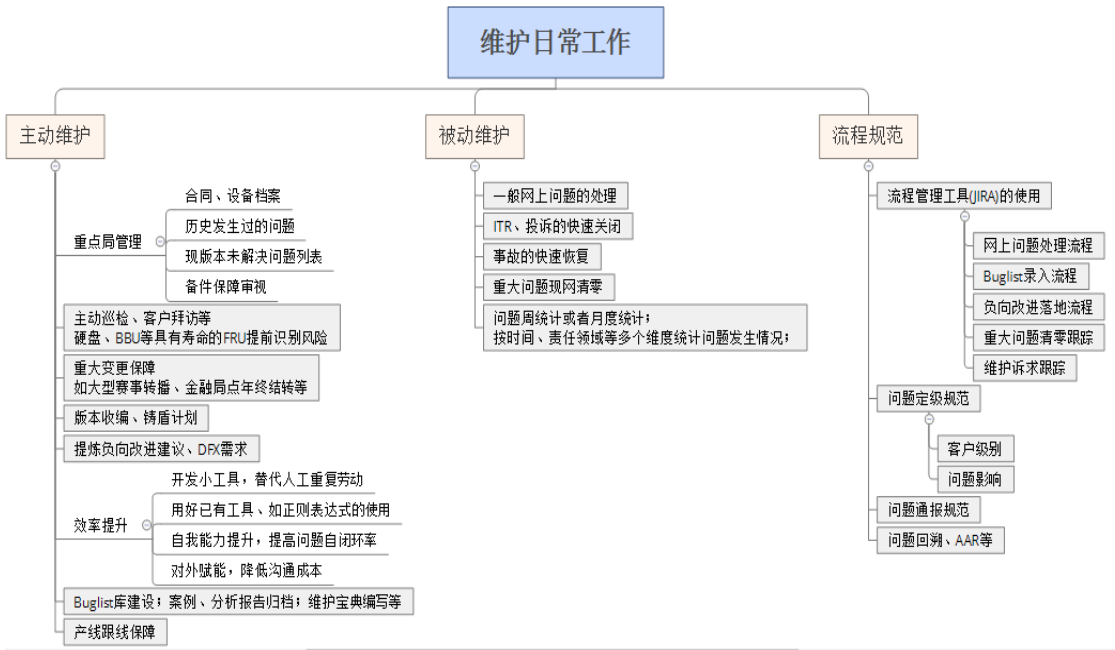
日常工作包含众多事项，每件事项具有多个环节，每个环节需要不同的角色来处理，怎样将**事项、环节、员工角色**三者系统地组织起来；既能有效的跟踪每项事务落地，又能最大程度降低沟通交流成本？

本文档以上市保障为例，介绍一种利用 workflow 软件管理日常工作的方法；除上市保障外，利用 workflow 软件管理其他日常工作的方法与之类似。

一、上市保障日常工作

上市保障工作开始意味着本轮开发工作基本结束，产品上市后的运行状况是对产品开发质量的一次检验；上市保障工作除及时解决客户现场暴出的问题外，一个重要的职责是从在网设备问题中提取避免问题重犯的准则和改进点，并促使这些准则和改进在下一轮的开发中落地以提升新一代产品质量。

- 上市保障工作可分为主动维护和被动维护两大方面，每个方面又可以细分为多项具体工作，如下所示：
- 被动维护：主要指问题发生后的响应和处理；如上市保障问题处理，事故的快速恢复，ITR、投诉的快速关闭，问题发生后的客户情绪安抚等。
 - 主动维护：主要指避免问题发生，提升产品质量、DFX 特性，提高客户满意度等；如针对金融、国家电网等客户设立重点局设备档案；阿里等重点客户在双十一等关键时间点的保障；避免处理问题过程中发现的低级错误重犯；高效地跟踪改进点落地；快速将典型问题的处理技能传递给轮转到上市保障组的同事等。



二、上市保障痛点

- 工作内容复杂，耗费大量精力依然无序。
- 被问题攻关搞得焦头烂额，彻夜不眠，最后发现是一个低级问题
- 屡次被客户质疑，因为产品质量羞愧难当。
- 解决这些问题的过程中有很多闪亮的点子，一些可供后续设计、开发时参考进而避免重蹈覆辙的规范，可惜没有积累下来，新一代产品开发时又在问题的泥潭中苦苦挣扎。

怎样系统有序的管理各项工作？怎样避免初级问题重犯？上市保障的典型问题如何积累下来，怎样对上市保障组的新组员进行快速赋能？

三、解决方案

- 1.梳理日常工作，将例行工作划分为多个业务流程，如问题处理流程，改进流程等；

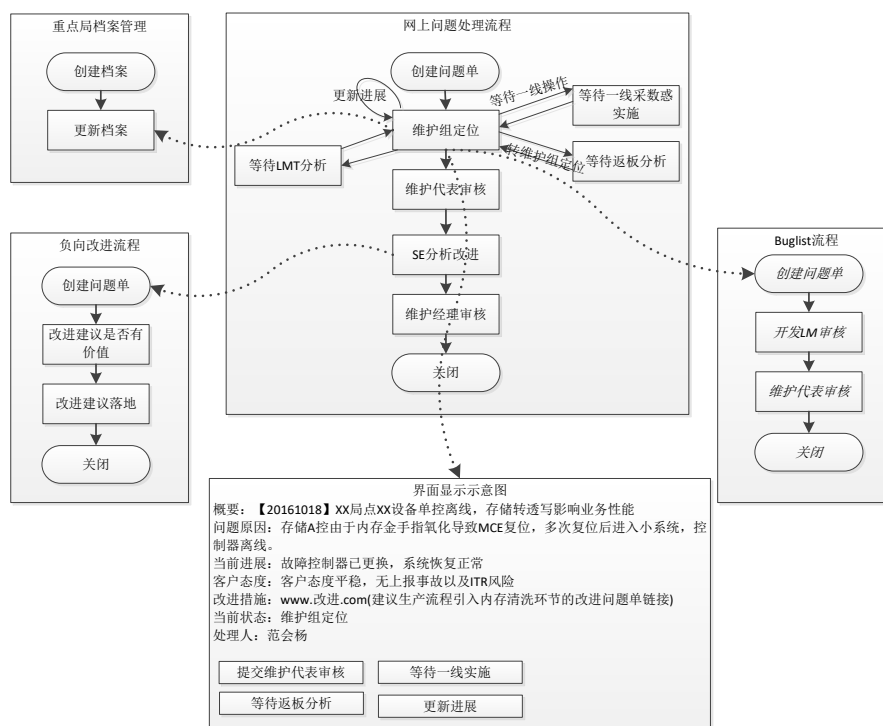
- 2.每个业务划分多个环节，并为每个环节指定处理人；
- 3.规划好以后由专人利用业务流程管理(BPM)工具进行实施；
- 4.工具实施后，各角色作为用户登录系统，处理分配给自己的问题单，处理完成后将问题单走给下个环节处理人即可；如问题处理流程中问题处理人完成问题分析，维护代表完成分析审核，SE 完成改进分析并创建改进问题单等。
- 在处理问题的过程中，完成典型问题的提取、优化改进的提取等工作，后续可以登录系统查看跟踪优化改进的落地，也可以查看归档的典型问题。

TaskSteper 是基于 django 框架实现的建议的 BPM 工具，可以协助各任务角色高效地分工协作。

四、工具使用示例

1.流程梳理

将上市保障组的日常工作分为网上问题处理、重点局档案管理、负向改进、典型 Buglist 管理等几个流程；并为每个流程划分为多个环节，以流程确保被跟踪的事务闭环，如下所示：



2.事务跟踪在 TaskSteper 系统中的实现(管理员部分)

该部分属于简单的二次开发，部门中有一个人熟悉工具即可，以网上处理流程为例，步骤如下：

2.1)定义字段 Model

此处定义为 IssueTraceModel，一个 Model 对应数据库的一张表，Model 的一个 field，如 summary、detail 等对应表中的一列。

```
class IssueTraceModel(models.Model):
    summary = models.CharField(max_length=200, verbose_name="概要", default="【20170101】XX局点XX设备发生XX问题")
    detail = models.CharField(max_length=5000, verbose_name="详细描述")
    current_process = models.CharField(max_length=5000, verbose_name="处理进展", default="##由维护处理人填写##")
    issue_level = models.CharField(max_length=200, verbose_name="问题级别", choices=CHOICE_ISSUE_LEVEL)
    customer_type = models.CharField(max_length=200, verbose_name="客户类别", choices=CHOICE_CUSTOMER_TYPE)
    class_1 = models.CharField(max_length=200, verbose_name="问题大类", choices=CHOICE_ISSUE_CLASS_1)
    class_2 = models.CharField(max_length=200, verbose_name="问题小类", choices=CHOICE_ISSUE_CLASS_2)
    class_3 = models.CharField(max_length=200, verbose_name="责任领域", choices=RESPONSE_FIELD_CHOICES)
    buglist_url = models.CharField(max_length=200, default="##Buglist单链接(若有)##", verbose_name="Buglist链接(URL)")
    improvement_url = models.CharField(max_length=200, default="##改进单链接(若有)##", verbose_name="改进项链接(URL)")
    issue_processor = models.CharField(max_length=200, default="##提交人填写##", verbose_name="维护处理人")
    issue_checker = models.CharField(max_length=200, default="##维护处理人指定##", verbose_name="维护代表")
    issue_se = models.CharField(max_length=200, default="##维护代表指定##", verbose_name="改进提取SE")
    assigned_to = models.CharField(max_length=200, default="", verbose_name="当前处理人(只读)")
    created_by = models.CharField(max_length=200, default="", verbose_name="创建人(只读)")
    current_state = models.CharField(max_length=200, verbose_name="当前状态(只读)")
```

Field 的属性说明如下：

max_length	字段的最大长度
verbose_name	字段在界面的显示名称，见下图
default	字段默认值
choices	下拉型字段的列表，class_1 字段的 choices 如右图所示，需在 IssueTrackModel 之前定义。

```
CHOICE_ISSUE_CLASS_1 = (
    ('软件', '软件'),
    ('硬件', '硬件'),
    ('结构', '结构'),
    ('误操作', '误操作'),
    ('配置', '配置'),
)
```

2.2) 定义字段 Form

Form 对应于界面显示一个表单，fields 属性决定 Model 中的哪些字段会显示在表单中（‘__all__’ 表示全部显示），widgets 主要定义了字段在显示时的一些属性，如显示的大小，是否只读等。此外需要将 Form 的 model 属性设置成步骤 2.1 中的 model。

```
class IssueTrackForm(ModelForm):
    class Meta:
        model = IssueTrackModel
        fields = '__all__'
        widgets = {
            'summary': forms.TextInput(attrs={'size': 79}),
            'current_process': forms.Textarea(attrs={'cols': 80, 'rows': 5}),
            # 'viewer_advice': forms.Textarea(attrs={'cols': 80, 'rows': 5}),
            'detail': forms.Textarea(attrs={'cols': 80, 'rows': 5}),
            'buglist_url': forms.TextInput(attrs={'size': 79}),
            'improvement_url': forms.TextInput(attrs={'size': 79}),
            'assigned_to': forms.TextInput(attrs={'readonly': True, 'size': 20}),
            'created_by': forms.TextInput(attrs={'readonly': True}),
            'current_state': forms.TextInput(attrs={'readonly': True}),
        }
```

2.3) 将上述两个步骤定义的 Model 和 Form 添加到 FormAndModelDict 中。

TaskSteper 支持创建多个项目，每个项目有独立的流程和字段，FormAndModelDict 中‘issue_track’为项目名，‘网上问题处理’为网页上显示的项目中文名。PrjModelClass 和 PrjFormClass 分别指定每个项目使用的 Model 和 Form 类。

```
PRJ_NAME_LIST = ['improvement', 'device_card', 'issue_track']

FormAndModelDict = {
    'improvement': {'PrjNameZh': '改进建议', 'PrjModelClass': NameModel, 'PrjFormClass': NameForm},
    'device_card': {'PrjNameZh': '设备管理', 'PrjModelClass': DeviceCardModel, 'PrjFormClass': DeviceCardForm},
    'issue_track': {'PrjNameZh': '网上问题处理', 'PrjModelClass': IssueTrackModel, 'PrjFormClass': IssueTrackForm},
}
```

2.4) 将定义好的 model 迁移到数据库中

在 django 工程目录下执行以下两条命令即可：

python manage.py makemigrations;

python manage.py migrate;

迁移成功后会看到类似如下提示（后补示意截图，其中的 model 名字与上文要增加的 model 不一致，可不关注）：

```
(venv) root@iZ2ze0ybhzh8v03jrk4kd0Z:~/virenv_python3/django_for_study/mysite# python manage.py makemigrations;
Migrations for 'polls':
  polls/migrations/0010_verreleasemodel.py
    - Create model VerReleaseModel
(venv) root@iZ2ze0ybhzh8v03jrk4kd0Z:~/virenv_python3/django_for_study/mysite# python manage.py migrate;
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying polls.0010_verreleasemodel... OK
```

迁移完成后可以看到数据库中已经创建了与 Model 对应的数据库表：

```
mysql> describe polls_issuetrackmodel;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)   | NO   | PRI | NULL    | auto_increment |
| summary    | varchar(200) | NO   |     | NULL    |              |
| detail     | varchar(5000) | NO   |     | NULL    |              |
| current_process | varchar(5000) | NO   |     | NULL    |              |
| issue_level | varchar(200) | NO   |     | NULL    |              |
| customer_type | varchar(200) | NO   |     | NULL    |              |
| class_1    | varchar(200) | NO   |     | NULL    |              |
| class_2    | varchar(200) | NO   |     | NULL    |              |
| class_3    | varchar(200) | NO   |     | NULL    |              |
| buglist_url | varchar(200) | NO   |     | NULL    |              |
| improvement_url | varchar(200) | NO   |     | NULL    |              |
| issue_processor | varchar(200) | NO   |     | NULL    |              |
| issue_checker | varchar(200) | NO   |     | NULL    |              |
| issue_se   | varchar(200) | NO   |     | NULL    |              |
| assigned_to | varchar(200) | NO   |     | NULL    |              |
| created_by | varchar(200) | NO   |     | NULL    |              |
| curent_state | varchar(200) | NO   |     | NULL    |              |
+-----+-----+-----+-----+-----+-----+
17 rows in set (0.00 sec)
```

通过 uwsgi 启动 django 服务，可以看到定义完成后界面显示效果如下所示：

当前项目:

网上问题处理

【概要:】

【20170101】XX局点XX设备发生XX问题

【详细描述:】

【问题现象】: xx

【问题影响】: xx

【客户诉求】: xx

【客户态度】: xx

【处理进展:】

##由维护处理人填写##

【问题级别:】

一般问题

【客户类别:】

金融

【问题大类:】

硬件

【问题小类:】

线缆

【责任领域:】

线缆

【Buglist链接(URL):】

#Buglist单链接(若有)#

【改进项链接(URL):】

#改进单链接(若有)#

【维护处理人:】

#提交人填写#

【维护代表:】

#维护处理人指定#

【改进提取SE:】

#维护代表指定#

【当前处理人(只读):】

fanhuiyang

【创建人(只读):】

fanhuiyang

【当前状态(只读):】

维护人员处理

创建

• [首页](#)

2.5) 定义状态转换表 FSM_TRANS_ISSUE_TRACK

```
#上市保障问题跟踪
issue_track_action_1 = {'assign_to':'维护代表'}
issue_track_action_2 = {'assign_to':'维护处理人'}
issue_track_action_3 = {'assign_to':'改进提取SE'}
FSM_TRANS_ISSUE_TRACK = [
    {'source': '维护人员处理', 'trigger': '更新进展', 'dest': '维护人员处理', 'trans_condition': {}, 'trans_action': {'assign_to': '创建人(只读)' }},
    {'source': '维护人员处理', 'trigger': '提交审核', 'dest': '维护代表审核', 'trans_condition': {}, 'trans_action': issue_track_action_1},
    {'source': '维护代表审核', 'trigger': '打回补充信息', 'dest': '维护人员处理', 'trans_condition': {}, 'trans_action': issue_track_action_2},
    {'source': '维护代表审核', 'trigger': '转SE分析改进', 'dest': 'SE分析改进', 'trans_condition': {}, 'trans_action': issue_track_action_3},
    {'source': 'SE分析改进', 'trigger': '问题关闭', 'dest': '关闭', 'trans_condition': {}, 'trans_action': {}}
```

此处一个状态对应日常工作的一个环节，表中每一行表示一个状态转换，包括以下元素：

source	源状态
trigger	对应界面上的触发按钮，点击按钮时会将问题单转到目标状态
dest	目标状态
trans_condition	转换条件，只有满足转换条件才会生成相应的按钮
trans_action	转换时的操作，目前支持 assign_to（将问题分配给某个角色）等

定义完成后将项目名称和对应的状态转换表添加到 TRANS_TABLE_DICT 中：

```
#定义各个项目对应的状态转换表
TRANS_TABLE_DICT = {
    'improvement':FSM_TRANS_TABLE,
    'device_card':FSM_TRANS_TABLE_DEVICECARD,
    'issue_track':FSM_TRANS_ISSUE_TRACK
}
```

3.用户使用 TaskSteper 跟踪问题处理（用户部分）

具体到一个问题的处理，流程如下所示

- 维护人员接收到问题后创建条目，录入问题单库，记录时间、局点、设备型号、问题描述等信息；

• 当前项目：网上问题处理

• [创建条目](#)

• [所有条目](#)

• [返回首页](#)

待处理条目：

ID	概要	当前状态	当前处理人	报告人
question_1	【20170101】XX局点XX设备发生XX问题	SE分析改进	fanhuiyang	fanhuiyang
question_2	【20170101】XX局点XX设备发生XX问题	维护代表审核	fanhuiyang	fanhuiyang
question_3	【20170101】XX局点XX设备发生XX问题	维护人员处理	fanhuiyang	fanhuiyang

当前项目:

网上问题处理

【概要:】

【20170101】XX局点XX设备发生XX问题

【详细描述:】

【问题现象】: xx

【问题影响】: xx

【客户诉求】: xx

【客户态度】: xx

##由维护处理人填写##

【处理进展:】

【问题级别:】

一般问题

【客户类别:】

金融

【问题大类:】

硬件

【问题小类:】

线缆

【责任领域:】

线缆

【Buglist链接(URL):】

#Buglist单链接(若有)#

【改进项链接(URL):】

#改进单链接(若有)#

【维护处理人:】

#提交人填写#

【维护代表:】

#维护处理人指定#

【改进提取SE:】

#维护代表指定#

【当前处理人(只读):】

fanhuiyang

【创建人(只读):】

fanhuiyang

【当前状态(只读):】

维护人员处理

创建

• [首页](#)

- 在“维护人员处理”阶段，问题处理人补充问题信息，更新问题进展；如果该问题是多次出现的典型问题，需要在“buglist”管理项目中增加一个问题，并将问题地址填入此处的“Buglist 连接(URL)”字段中。处理完成后填入维护代表，点击“提交审核”将问题单提交审核，问题转给下个角色（维护代表）处理。

【当前项目】 网上问题处理 【ID】 question_3

【概要:】

【20170101】XX局点XX设备发生XX问题

【详细描述:】

【问题现象】: xxx

【问题影响】: xx

【客户诉求】: xx

【客户态度】: xx

【处理进展:】

处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展
处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展
处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展
处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展
处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展处理进展

【问题级别:】

一般问题

【客户类别:】

金融

【问题大类:】

硬件

【问题小类:】

线缆

【责任领域:】

线缆

【Buglist链接(URL):】

#Buglist单链接(若有)#

【改进项链接(URL):】

http://47.95.215.176:9000/polls/0/flow/improvement/2/detail/

【维护处理人:】

#提交人填写#

【维护代表:】

#维护处理人指定#

【改进提取SE:】

#维护代表指定#

【当前处理人(只读):】

fanhuiyang

【创建人(只读):】

fanhuiyang

【当前状态(只读):】

维护人员处理

更新进展

提交审核

• [返回首页](#)

• [返回项目首页](#)

- SE 分析改进阶段，若从该问题中发现需改进点，则创建一个改进单用于跟踪改进落地，并将改进单的地址填入“改进项链接(URL)”字段。

【当前项目】网上问题处理【ID】question_1

【概要:】

【20170101】XX局点XX设备发生XX问题

问题影响，客户态度等

【详细描述:】

##由维护处理人填写##

【处理进展:】

【问题级别:】

一般问题

【客户类别:】

煤炭

【问题大类:】

配置

【问题小类:】

误操作

【责任领域:】

误操作

【Buglist链接(URL):】

#Buglist单链接(若有)#

【改进项链接(URL):】

http://47.95.215.176:9000/polls/0/flow/improvement/2/detail/

【维护处理人:】

#提交人填写#

【维护代表:】

fanhuiyang

【改进提取SE:】

fanhuiyang

【当前处理人(只读):】

fanhuiyang

【创建人(只读):】

fanhuiyang

【当前状态(只读):】

SE分析改进

问题关闭

• [返回首页](#)

• [返回项目首页](#)

该方法的优势:

1) 极大降低沟通成本

要了解一项工作的进展，只需要登录系统找到表示该工作的问题单就能一目了然，避免了表格跟踪事务时所需的专人拉会收集信息、汇总、分发、各人更新等繁琐的步骤。

2) 有效跟踪事务闭环

各环节责任人明确，当前状态明确，可以有效地跟踪事务从开始到闭环的全过程。

3) 团队知识有序的积累和归档

日常工作按照上述方法开展，在问题处理过程中就可以完成典型问题的积累，改进点的提取、跟踪、落地等，并且在问题处理项目和 Buglist、改进流程之间通过 URL 链接建立联系，可以通过问题模块、发生局点等信息等快速找到相应问题单，进而找到积累的典型问题信息或者改进信息；也可以通过典型问题信息或者改进信息找到对应的网上问题单，了解问题发生背景；是一种较好的团队知识归档方法。

积累的改进建议列表示例:

• 当前项目: 改进建议

• [创建条目](#)

• [所有条目](#)

• [返回首页](#)

待处理条目:

ID	概要	当前状态	当前处理人	报告人
question_1	改进建议1改进建议1改进建议1改进建议1改进建议1	提交建议	fanhuiyang	fanhuiyang
question_2	改进建议2改进建议2改进建议2改进建议2改进建议2改进建议价值评审	价值评审	fanhuiyang	fanhuiyang

积累的改进项详细信息示例:

【当前项目】改进建议【ID】question_1

【概要:】

改进建议1改进建议1改进建议1改进建议1改进建议1

改进建议1改进建议1改进建议1改进建议1

【详细描述:】

##由改进实施人填写##

【处理进展:】

【改进价值评估:】

0

【改进建议评审人:】

#提交人填写#

【改进建议实施人:】

#由建议评审人指派#

【当前处理人(只读):】

fanhuiyang

【创建人(只读):】

fanhuiyang

【当前状态(只读):】

提交建议

提交评审

• [返回首页](#)

• [返回项目首页](#)

五、进一步优化

增加按字段搜索、统计、导出等功能。

例如现在每次发版本需要手动写 **release notes**，实际上可以增加“**release notes**”字段，每次发版本时，根据问题单走到解决状态的时间、技术小组等过滤出这一版本解决问题，然后将这些问题的“**release notes**”字段导出即可。