# Bug总结报告

## 目的/背景：

通过统计分析定位到bug率较高的具体因素，在制定对应的质量管理措施，以提高研发质量，降低开发成本，缩短开发周期，提高自测bug发现效率，降低bug数量，最终达成今年的产品年度BUG下降率大于20%

## 二、调查/分析方法：

提取QA对VDI5.3.0版本的bug统计数据为样本，对bug进行分布情况分析，实现难度情况分析等

**分析方法：**

对bug在对应需求上的严重程度情况、bug类型的分布情况、实现难度分布情况进行汇总统计。

1. **Bug需求分布与严重程度分布情况**

对bug的严重程度分类汇总，类别分为：二级、三级、四级。

同时将bug匹配上对应的需求，分析出bug数较高的需求，供开展原因分析。

1. **Bug类型分布情况维度介绍**

* **需求因素**：需求编写的不够全面、不完整、不准确、变更沟通不顺畅、变更需求录入的不及时。

类别分为：设计缺陷。

* **编码因素**：研发过程中的需求变更；开发人员对业务上的不理解或理解不一致；代码编写不严谨；测试用例覆盖遗漏。

类别分为：代码错误、需求未实现、代码改进、异常路径、业务不熟悉、修改引入

* **其他因素**：类别分为：配置相关、性能问题、其他无法分类的bug。

|  |  |  |
| --- | --- | --- |
| **因素** | **Bug类型** | **概念** |
| 需求因素 | 设计缺陷 | 即需求设计不合理 |
| 需求不明确 | 需求编写的不够全面、不完整、不准确 |
| 需求变更 | 变更沟通不顺畅、变更需求录入的不及时 |
| 编码因素 | 代码错误 | 程序异常的问题 |
| 代码改进 | 代码编写不严谨出现的BUG |
| 需求未实现 | 程序未按需求实现 |
| 异常路径 | 异常路径考虑遗漏 |
| 界面优化 | 界面存在错别字、乱码、不美观、信息提示 |
| 业务不熟 | 对模块不熟悉或对依赖模块考虑不全 |
| 修改引入 | 修改老代码引入新bug |
| 其他人修改代码连锁造成自己模块的BUG |  |
| 其他因素 | 配置相关 |  |
| 性能问题 |  |
| 代码合并 |  |
| 建议 | 建议性的bug |
| 其他 | 暂不好分类的bug |

1. **Bug实现难易分布情况维度介绍**

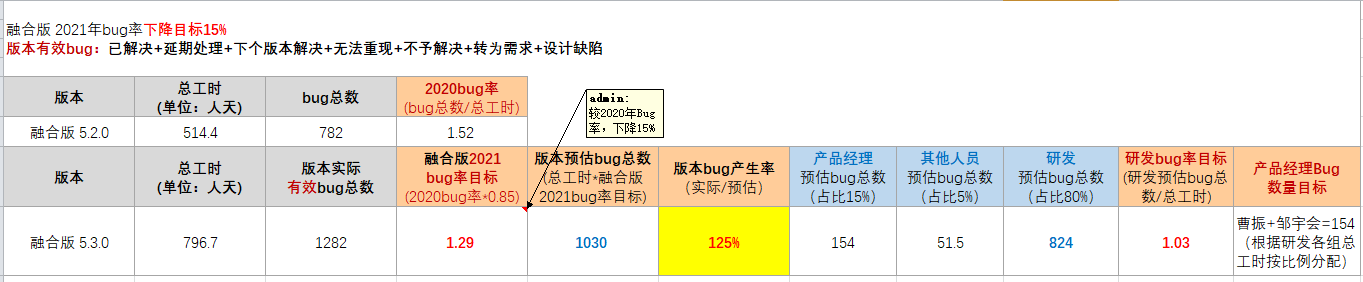
Bug引起原因从需求实现难易程度来分类：如下图

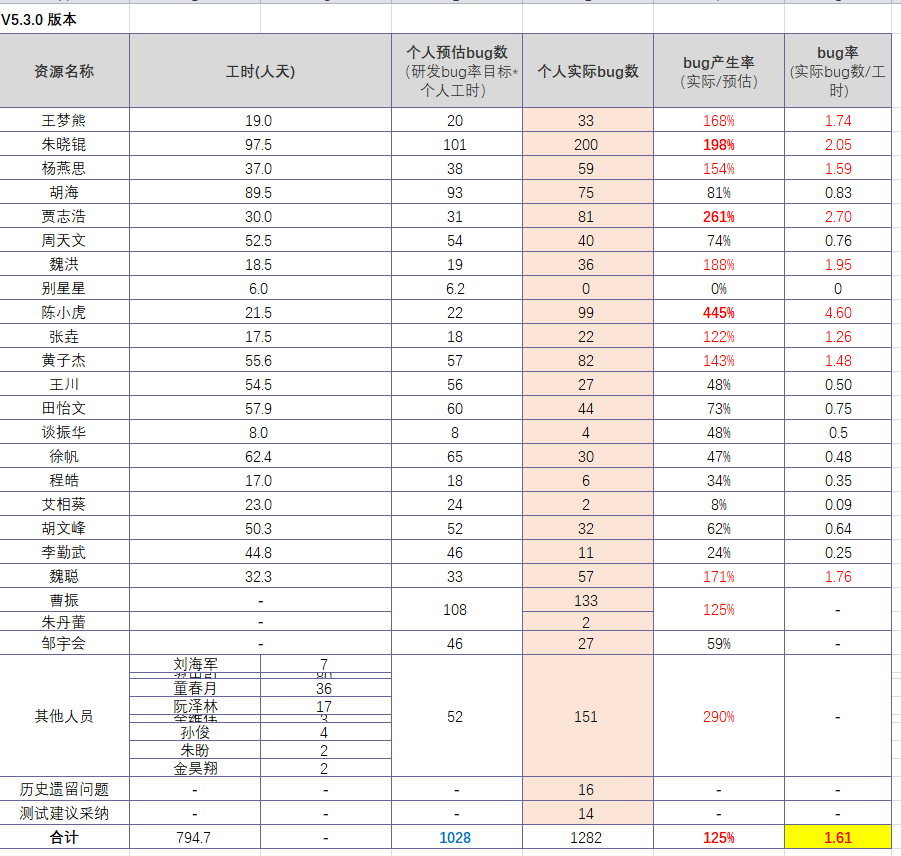
|  |  |
| --- | --- |
| **实现难度分类** | **概念** |
| 与技术有关 | 技术实现较复杂 |
| 业务场景较复杂 | 业务场景较复杂，异常路径较难模拟。 |
| 经常出现、易犯的**BUG** | 文案提示类型，空指针类型等 |

## 三、Bug统计数据分析/结论

VDI5.3.0 bug统计数据汇总如下

融合版 5.3.0 Bug率统计表如下：





### Bug需求分布与严重程度分布

表1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 服务层/客户端  [bug总数：403] | | 前端  [bug总数：144] | |
| Bug级别 | bug数 | 占比 | bug数 | 占比 |
| 二级 | 89 | 22.08% | 13 | 9.03% |
| 三级 | 248 | 61.54% | 106 | 73.61% **↑** |
| 四级 | 14 | 3.47% | 18 | 12.50% |

计算公式：占比=某级别bug数/bug总数

其中任务队列的bug总数：83个 占总数的20.6%

### Bug类型分布情况

表2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 服务层/客户端  [bug总数：403] | | 前端  [bug总数：144] | |
| 类型 | bug数 | 占比 | bug数 | 占比 |
| 编码因素 | 256 | 63.52% ↑ | 91 | 63.19% ↑ |
| 需求因素 | 77 | 19.11% | 41 | 28.47% |
| 其他因素 | 67 | 16.38% | 12 | 8.33% |

计算公式：占比=某类型因素bug数/bug总数

表2.2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 服务层/客户端  [bug总数：403] | | | 汇总 | 前端  [bug总数：144] | | | 汇总 |
| 类型 | bug数 | 占比 | 类型 | bug数 | 占比 |
| 设计缺陷 | 77 | 19.11% | **50.87 %** ↑ | 设计缺陷 | 41 | 28.47% | 61.8 % ↑ |
| 代码错误 | 69 | 17.12% | 代码错误 | 23 | 15.97% |
| 代码改进 | 59 | 14.64% | 代码改进 | 25 | 17.36% |
| 异常路径 | 37 | 9.18% | 17.86 % | 界面优化 | 15 | 10.42% | 18.75 % |
| 需求未实现 | 35 | 8.68% | 业务不熟 | 12 | 8.33% |
| 修改引入 | 25 | 6.20% | 30.27% | 需求未实现 | 9 | 6.25% | 19.43 % |
| 界面优化 | 21 | 5.21% | 修改引入 | 5 | 3.47% |
| 业务不熟 | 10 | 2.48% | 异常路径 | 2 | 1.39% |
| 性能问题 | 12 | 2.98% | 配置相关 | 1 | 0.69% |
| 配置相关 | 6 | 1.49% | 性能问题 | 1 | 0.69% |
| 其他（无法分类） | 48 | 11.91% | 其他（无法分类） | 10 | 6.94% |

计算公式：占比=某类型bug数/bug总数

### 实现难度分布情况

表3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 服务层/客户端  [bug总数：403] | | 前端  [bug总数：144] | |
| 实现难度 | bug数 | 占比 | bug数 | 占比 |
| 与技术有关 | 38 | 9.43% | 10 | 6.94% |
| 业务场景较复杂 | 122 | 30.27% ↑ | 34 | 23.61% |
| **经常出现、易犯的Bug** | **116** | **28.78%** ↑ | **66** | **45.83%** |
| 其他 |  |  |  |  |

计算公式：占比=某类型bug数/bug总数

### 预估可减少的Bug分布情况

表4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 服务层/客户端  [bug总数：403] | | 前端  [bug总数：144] | |
| 预估可减少的Bug类型 | bug数 | 占比 | bug数 | 占比 |
| 可能防止bug数 | 155 | 38.46% | 52 | 36.11% |
| 可能减少bug出现概率bug数 | 72 | 17.87% | 42 | 29.17% |
| 可以提前预警bug数 | 83 | 20.60% | 11 | 7.64% |
| 其他 |  |  |  |  |

上述维度【1】~【4】数据统计的结果显示（以服务端为例说明）：

在VDI5.3.0开发完成需求中，任务队列的bug总数：83个 占总数的20.6%。

编码因素、需求因素的引起的bug占总数的82%之多，主要集中在二、三级Bug，Bug绝大部分分布在设计缺陷、代码错误、代码改进项占总数的51%,异常路径、需求未实现项占总数18%，修改引入、界面优化占总数的11%。

其中大部分缺陷引起原因与对应需求实现技术难度关系不大仅占比9%，与业务场景复杂度和自测覆盖程度有直接关系占比总计达58%（不完全统计）。

绝大部分的bug是完全可以通过规范质量管理制度，并对产品实现过程进行有效监督来降低或者规避bug的产生。

## 四、复盘&总结的汇总

在大家的bug总结文档中提到很多问题，大致有如下几点：

1.需求设计缺陷、异常路径考虑不全或缺少异常路径的详细描述。

2.需求疑问的活动开展的不及时、不到位。

3.代码错误、测试路径遗漏。

4.代码冗余度较多的模块，修改时容易出错。

5.代码逻辑、语法、类方法等的错误使用引入的问题，缺少工具来检查代码。

6.预防体系不够完善，主要体现在产品质量自查大都靠个人，测试和开发是完全割裂，工作内容没有互通，经验教训没有共享，没有沉淀。

## 五、重点改进项、措施

针对bug占比较高的项，**代码错误和需求设计缺陷**两项暂定为重点改进项，讨论预防改进措施如下。

* 需求层面：

1. 补从禅道需求内容项（\*）

具体措施：补充需求包含[异常说明]项,异常路径较复杂时可以用图说明（如：状态图），如果隐性属性较多，异常情况说明较困难，可以在研发预研后一起补充。前提是要有这项，在设计、开发过程中不断完善，需将此项纳入到评审的标准中。

1. 规范需求疑问活动开展（\*）

《项目过程规范》中已经明确了设计阶段需包含《提出设计/测试需求疑问》的活动，该活动需**及时**开展且明确过程输出《需求设计/测试问题列表》。

具体措施：对较复杂的需求需明确将该活动写入开发、测试计划中，明确输出内容，QA在检查时作为指标项，如不开展该活动需备注原因、为此带来的风险、风险承受的能力。

* 研发层面：

1. 规范自测用例格式、审核流程、监督活动的开展（\*）

具体措施：

* 对复杂需求将活动写入开发计划，按计划开展审核。
* 统一自测用例的格式，区分首次自测范围和回归自测范围。
* 自测用例中须明确说明异常路径，异常情况比较复杂时研发可以开展异常设计评审
* **自测用例文档编写建议在编码完成前。**

1. 建立研发、测试沟通、分享的渠道。

具体措施：

沟通渠道建议主要采用复杂的需求互相评审的方式开展。

具体如下:

研发：对复杂的需求须开展自测用例评审（测试、产品需要参与）。

测试：对复杂的需求可以开展测试点、测试用例的评审（相关开发、产品需要参与）。

通过互相评审的方式让开发、测试、产品保持及时的良好的沟通，提高需求质量和对需求理解的一致性。

* 辅助措施：

1. 建议搭建代码检查的工具，对语法类问题在代码提交时能自动检测，建议复杂的接口需编写测试用例，尽量完善接口的测试用例覆盖范围。
2. 建议完善研发知识库，提高项目组成员对产品的技术水平、业务水平，将知识库开放给测试。

具体措施：建议采用不定期的文章分享，每月至少输出一篇技术文章，对于分享主题和内容可以通过计划分解至每个季度或者每个月。

1. 提高bug数据统计的准确性，提高对禅道操作的规范性。

具体措施：

1. Bug所属模块的规范
2. Bug解决方案规范（特别注意重复bug处理方案）
3. Bug类型分类规范

以上措施可以讨论优选几个针对性强的执行，执行一段周期后，在收集一次数据，看看执行前后的效果，验证bug率是否下降，bug率下降，开发成本是否也会下降。

## 六、关键风险：

1、需要需求、测试、研发三方面都需要开展工作的一些措施，如果时间上契合度不高，执行后结果可能与预期的存在较大偏差。