项目名称：MiniIDE – C语言编辑器

文档编号：1.0

MiniIDE – C语言编辑器

【软件测试方案】

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 总页数 |  | 正文 |  | 附录 |  | 生效日期 |  |
| 编制 |  | | | 批准 |  | | |

目录

[1 引言 4](#_Toc19131306)

[1.1 项目背景 4](#_Toc19131307)

[1.2 编写目的 4](#_Toc19131308)

[1.3 预期读者 4](#_Toc19131309)

[1.4 环境配置 4](#_Toc19131310)

[1.4.1 开发环境 4](#_Toc19131311)

[1.4.2 测试环境 5](#_Toc19131312)

[1.5 参考资料 5](#_Toc19131313)

[1.5.1 术语定义 5](#_Toc19131314)

[1.5.2 参考文档 6](#_Toc19131315)

[2 测试范围 7](#_Toc19131316)

[2.1 系统结构图 7](#_Toc19131317)

[2.2 系统功能表 7](#_Toc19131318)

[2.3 系统性能需求 8](#_Toc19131319)

[2.3.1 响应时间 8](#_Toc19131320)

[2.3.2 易用性 8](#_Toc19131321)

[2.3.3 可扩展性 8](#_Toc19131322)

[2.3.4 健壮性 9](#_Toc19131323)

[3 测试策略 10](#_Toc19131324)

[3.1 功能测试 10](#_Toc19131325)

[3.2 用户界面测试 10](#_Toc19131326)

[3.3 性能测试 10](#_Toc19131327)

[3.4 压力测试 11](#_Toc19131328)

[3.5 强度测试 11](#_Toc19131329)

[3.6 容量测试 11](#_Toc19131330)

[3.7 兼容性测试 11](#_Toc19131331)

[3.8 配置测试 12](#_Toc19131332)

[3.9 错误处理测试 12](#_Toc19131333)

[4 资源 13](#_Toc19131334)

[4.1 测试资源 13](#_Toc19131335)

[4.2 测试环境 13](#_Toc19131336)

[5 进度安排 14](#_Toc19131337)

[5.1 测试工作安排 14](#_Toc19131338)

[5.2 输出文档 14](#_Toc19131339)

[5.3 测试用例模板 15](#_Toc19131340)

[5.4 测试功能点矩阵 15](#_Toc19131341)

[5.5 测试性能点矩阵 17](#_Toc19131342)

[6 执行测试工作流程 18](#_Toc19131343)

[6.1 测试工作总体流程 18](#_Toc19131344)

[6.2 单元测试工作流程 18](#_Toc19131345)

[6.3 集成测试工作流程 19](#_Toc19131346)

[6.4 系统测试工作流程 19](#_Toc19131347)

[6.4.1 业务测试工作流程 19](#_Toc19131348)

[6.4.2 压力测试工作流程 20](#_Toc19131349)

[6.4.3 性能测试工作流程 20](#_Toc19131350)

[6.4.4 安装测试工作流程 21](#_Toc19131351)

[6.4.5 验收测试工作流程 21](#_Toc19131352)

[7 发布标准 22](#_Toc19131353)

[7.1 测试完成标准 22](#_Toc19131354)

[7.2 产品发布标准 22](#_Toc19131355)

[8 风险评估 23](#_Toc19131356)

[8.1 变更管理 23](#_Toc19131357)

[8.2 缺陷级别 24](#_Toc19131358)

[8.3 缺陷管理流程 25](#_Toc19131359)

[9 测试用例 26](#_Toc19131360)

[9.1 基础编辑 26](#_Toc19131361)

[9.2 编译运行 27](#_Toc19131362)

[9.2.1 测试样例1 27](#_Toc19131363)

[9.2.2 测试样例2 27](#_Toc19131364)

[9.3 多文件编译运行 28](#_Toc19131365)

[9.3.1 测试样例1 28](#_Toc19131366)

[9.3.2 测试样例2 28](#_Toc19131367)

[9.4 调试 30](#_Toc19131368)

[9.4.1 测试用例1 30](#_Toc19131369)

[9.4.2 测试用例2 30](#_Toc19131370)

[9.5 新需求——注释隐藏折叠 31](#_Toc19131371)

# 引言

## 项目背景

北京理工大学计算机学院软件工程专业软件工程实践课程。

为进一步提高学生的学术经验和实践能力，依照教学计划完成C语言编辑器开发平台。

## 编写目的

测试是软件工程中必不可少的一项工作，软件质量的保证必须通过测试实现。通过对本软件的测试，尽可能的发现软件中的错误，借以减少系统内部各模块的逻辑，功能上的缺陷和错误，保证每个单元能正确地实现其预期的功能。

为了提高检测出错误的几率，使测试能有计划地、有条不紊地进行，就必须要编制测试相关文件。而标准化的测试文件就如同一种通用的参照体系，可达到便于交流的目的。文件中所规定的内容可以作为对测试过程完备性的对照检查表，故采用这些文件将会提高测试过程的每个阶段的能见度，极大地提高测试工作的可管理性。

## 预期读者

本文档主要针对对象为软件管理人员、软件开发人员和软件测试人员。

## 环境配置

### 开发环境

1. 操作系统：Windows
2. 开发语言：C++ ， Qt
3. 开发平台：
4. Qt Creator
5. Qt版本信息：
6. Qt Creator 4.5.0(Community)；
7. Qt 5.10.0 for Desktop(MinGW 5.3.0 32 bit)；

### 测试环境

1. 操作系统：Windows
2. 开发平台：
3. Qt Creator
4. Qt版本信息：
5. Qt Creator 4.5.0(Community)；
6. Qt 5.10.0 for Desktop(MinGW 5.3.0 32 bit)；

## 参考资料

### 术语定义

1. Qt

面向对象的跨平台C++图形用户界面应用程序开发框架。

Qt跨平台C++图形用户界面应用程序开发框架。它既可以开发GUI程序，也可用于开发非GUI程序，比如控制台工具和服务器。Qt是面向对象的框架，使用特殊的代码生成扩展（称为元对象编译器(Meta Object Compiler, moc)）以及一些宏，Qt很容易扩展，并且允许真正地组件编程。

1. Qsciscintilla

Scintilla是一个免费、跨平台、支持语法高亮的编辑控件。它完整支持源代码的编辑和调试，包括语法高亮、错误指示、代码完成（code completion）和调用提示(call tips)。能包含标记（marker）的页边（margin）可用于标记断点、折叠和高亮当前行。而QScintilla是Scintilla在QT上的移植。

1. gcc / g++

GCC（GNU Compiler Collection，GNU编译器套件），是由 GNU 开发的编程语言编译器。它是以GPL许可证所发行的自由软件，也是 GNU计划的关键部分。GCC原本作为GNU操作系统的官方编译器，现已被大多数类Unix操作系统（如Linux、BSD、Mac OS X等）采纳为标准的编译器，GCC同样适用于微软的Windows。gcc 和 g++分别是GNU的C 和 C++ 的编译器。

1. gdb

GNU项目调试器GDB允许您在执行时查看另一个程序内部的内容 - 或者在崩溃时另一个程序正在执行的操作。 GDB可以做四种主要的事情（加上支持这些事情的其他事情）来帮助你捕捉行为中的错误：

1. 启动程序，指定可能影响其行为的任何事情。
2. 使程序在指定条件下停止。
3. 当程序停止时，检查发生了什么。
4. 更改程序中的内容，以便您可以尝试纠正一个错误的影响并继续了解另一个错误。

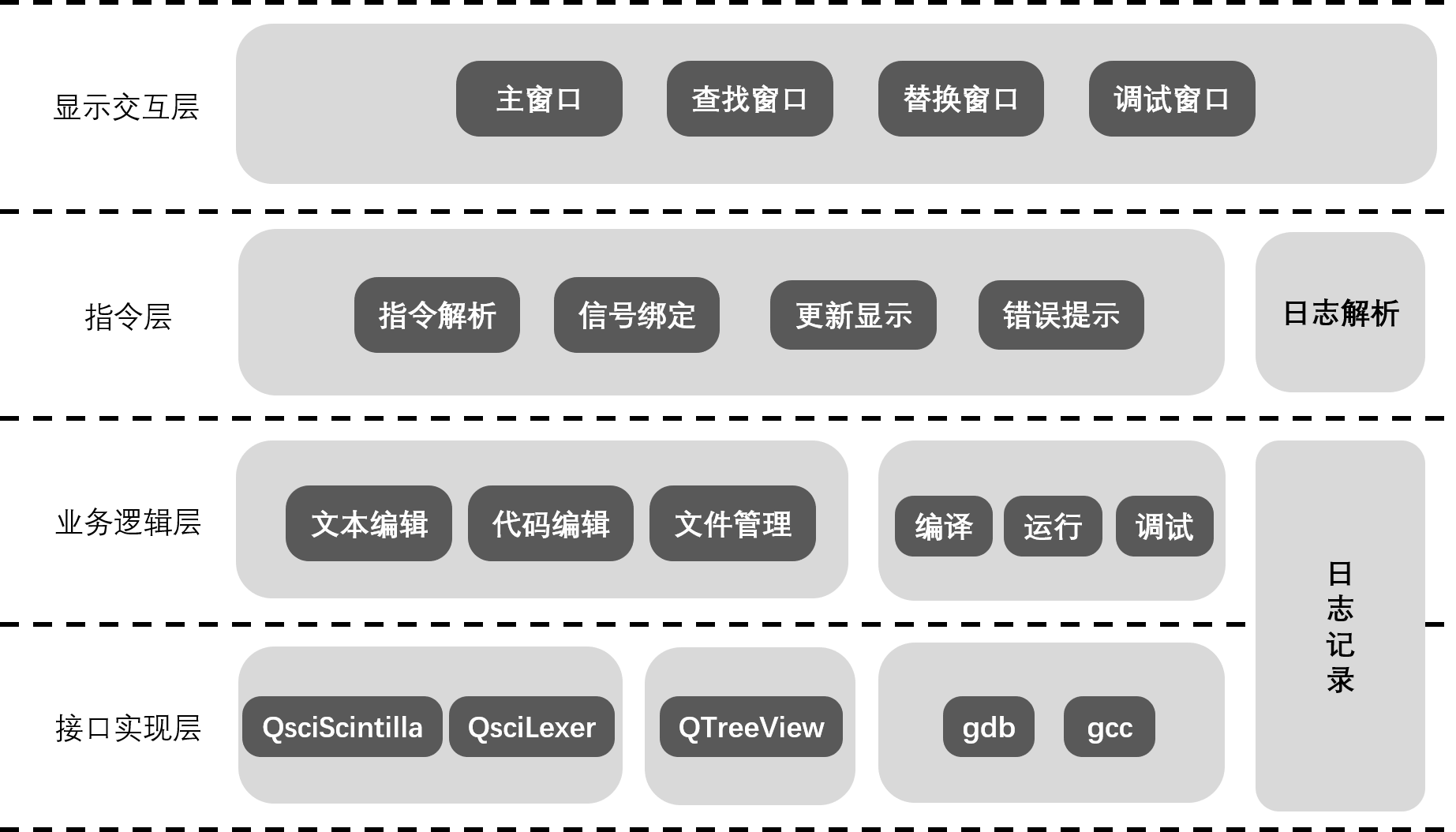
这些程序可能与GDB（本机），另一台机器（远程）或模拟器在同一台机器上执行。 GDB可以在大多数流行的UNIX和Microsoft Windows变体上运行，也可以在Mac OS X上运行。

### 参考文档

1. 软件测试方法和技术（第二版），朱少民主编，清华大学出版社
2. 软件测试方案模板

# 测试范围

## 系统结构图



## 系统功能表

|  |  |  |  |
| --- | --- | --- | --- |
| 程序功能表 | 文本编辑 | 复制 | Ctrl + C |
| 粘贴 | Ctrl + V |
| 查找 | Ctrl + F |
| 替换 | Ctrl + H |
| 撤销 | Ctrl + Z |
| 重做 | Ctrl + Y |
| 在状态栏显示当前行数 |  |
| 调整字体大小 | Ctrl + wheelEvent |
| 代码编辑 | 关键字识别高亮 |  |
| 关键字联想提示、补全 | Tab |
| 括号自动匹配高亮 |  |
| 括号自动补全 |  |
| 变量重命名 | F2 |
| 自动缩进 |  |
| 多行注释 | Ctrl + Shift + / |
| 代码块折叠 |  |
| 对已编辑的函数进行高亮显示 |  |
| 代码跳转 | Ctrl + mouseClick |
| 行格式排版 |  |
| 整体格式排版 |  |
| 注释显示 / 隐藏 | F8 |
| 代码运行 | 通过gcc对当前文件编译 | Ctrl + B |
| 通过gcc对当前文件运行 | Ctrl + R |
| 通过gcc实现多文件编译 | Ctrl + Shift + B |
| 通过gcc实现项目运行 | Alt + R |
| 在控制台显示编译结果 |  |
| 通过gdb对当前文件调试 | F5 |
| 文件管理 | 新建文件 | Ctrl + N |
| 打开文件 | Ctrl + O |
| 打开文件夹 | Ctrl + Shift + O |
| 保存文件 | Ctrl + S |
| 另存为文件 | Ctrl + shift + S |
| 多文件编辑 |  |
| 树形文件资源管理 |  |

## 系统性能需求

### 响应时间

系统应具有快速响应的特性：

1. 用户打开界面和提交事务的平均响应时间应低于1秒；
2. 用户进行Debug操作的响应时间应低于2s。

### 易用性

1. 目标系统用户界面应操作简洁、易用、灵活，风格统一易学。
2. 系统的用户帮助文档要求齐备，易于进行软件使用。
3. 系统通过Qt为用户提供GUI图形化界面拓展，并与使用普遍程度高的编辑器在图形界面、快捷键等功能设计一致，充分考虑系统的易用性。

### 可扩展性

本项目通过Qt开发，具有C++的可扩展性，能够根据用户的需求不断更新设计，进行功能拓展并预留接口，利于以后的升级与扩展。

### 健壮性

同时，用户在对系统进行操作时，由于各种原因会进行误操作，或者输入错误的数据等，系统应能够对这些情况进行处理，系统应保证个别模块出现问题不会对其他模块造成影响。

# 测试策略

## 功能测试

|  |  |
| --- | --- |
| 测试目标 | 确保功能测试需求项以及用例场景能够实现 |
| 测试范围 | 测试数据精确度、数据类型、业务功能等相关方面的正确性 |
| 测试技术 | 采用黑盒测试、白盒测试、边界测试以及等价类测试 |
| 工具和方法 | 手工测试 |
| 开始标准 | 开发阶段对应功能完成且测试用例设计完毕 |
| 完成标准 | 测试完所有功能且获得对应数据（用时等） |
| 特殊事项 | 无 |

## 用户界面测试

|  |  |
| --- | --- |
| 测试目标 | 测试图形化界面的对象能否正确反映业务的功能和需求 |
| 测试范围 | 测试数据精确度、数据类型、业务功能等相关方面的正确性 |
| 测试技术 | 为每个窗口创建或修改测试，以核实各个应用程序窗口和对象都可正确地进行浏览，并处于正常的对象状态。 |
| 工具和方法 | 手工测试 |
| 开始标准 | 开发阶段对应功能完成且测试用例设计完毕 |
| 完成标准 | 测试完所有功能且获得对应数据（用时等） |
| 特殊事项 | 无 |

## 性能测试

|  |  |
| --- | --- |
| 测试目标 | 针对系统的响应时间、易用性、可拓展性、健壮性等方面制定测试用例，分析结果确定系统性能。 |
| 测试范围 | 测试数据精确度、数据类型、业务功能等相关方面的正确性 |
| 测试技术 | 采用黑盒测试对每个不同性能进行测试。 |
| 工具和方法 | 手工测试 |
| 开始标准 | 开发阶段对应功能完成且测试用例设计完毕 |
| 完成标准 | 测试完所有功能且获得对应数据（用时等） |
| 特殊事项 | 无 |

## 压力测试

|  |  |
| --- | --- |
| 测试目标 | 得到性能指数最小时候（可以接受的最小指数）最大的压力数。 |
| 测试技术 | 黑盒测试、边界测试 |
| 完成标准 | 单个事务或单个用户：在每个事务所预期或要求的时间范围内成功地完成测试脚本，没有发生任何故障。  多个事务或多个用户：在可接受的时间范围内成功地完成测试脚本，没有发生任何故障。 |
| 特殊事项 | 无 |

## 强度测试

|  |  |
| --- | --- |
| 测试目标 | 在极端情况下出现的问题。 |
| 测试技术 | 在资源少和共享资源竞争的情况下发现可能的错误。  黑盒测试、边界测试 |
| 完成标准 | 在极端情况下依然可以正常运行。 |
| 特殊事项 | 无 |

## 容量测试

|  |  |
| --- | --- |
| 测试目标 | 使用大量数据给软件以考验，以确定达到限制时是否引发软件失败。 |
| 测试技术 | 1000行以上的代码量进行编辑、编译、运行等功能 |
| 完成标准 | 在输入大量数据的情况下，依然无重大问题发生。 |
| 特殊事项 | 无 |

## 兼容性测试

|  |  |
| --- | --- |
| 测试目标 | 在不同操作系统下，测试对象可正确安装到对应硬件配置中，并成功运行。 |
| 测试技术 | 黑盒测试 |
| 完成标准 | 测试对象可以成功运行且获得对应数据 |
| 特殊事项 | 无 |

## 配置测试

|  |  |
| --- | --- |
| 测试目标 | 确保本系统在有其他软件运行的情况下可以不受太大影响 |
| 测试技术 | 在运行本系统的同时打开其他软件，如：Dev，VS等 |
| 完成标准 | 打开其他程序时，本程序影响程度较低 |
| 特殊事项 | 无 |

## 错误处理测试

|  |  |
| --- | --- |
| 测试目标 | 在错误执行命令、输入非法数据或者强制终止测试对象时，检测测试对象报错纠错的能力和稳定性。 |
| 测试技术 | 黑盒测试，边界测试 |
| 完成标准 | 测试对象对错误信息反馈提示 |
| 特殊事项 | 无 |

# 资源

## 测试资源

|  |  |  |
| --- | --- | --- |
| 测试资源 | | |
| 角色 | 测试人员 | 具体职责 |
| 测试项目经理 | 张佳明  朱长昊 | 1. 提供技术指导 2. 明确测试要点、要求 3. 撰写测试方案 4. 编写测试用例 |
| 测试人员 | 刘震宇  湛蓝蓝  蒋雨彤 | 1. 执行测试 2. 记录结果 3. 从错误中恢复 4. 记录变更请求 |

## 测试环境

|  |  |  |
| --- | --- | --- |
| 系统资源 | | |
| 资源 | 名称 | |
| 操作系统 | Windows 10 | |
| 测试平台 | Qt Creator | Qt Creator 4.5.0(Community)； |
| Qt 5.10.0 for Desktop(MinGW 5.3.0 32 bit)； |

# 进度安排

## 测试工作安排

|  |  |  |  |
| --- | --- | --- | --- |
| 任务 | 工作量 | 开始日期 | 结束日期 |
| 制定测试计划 | 1个工作日 | 2019/9/9 | 2019/9/9 |
| 设计测试 | 1个工作日 | 2019/9/9 | 2019/9/9 |
| 实施测试 | 1个工作日 | 2019/9/10 | 2019/9/10 |
| 执行测试 | 1个工作日 | 2019/9/10 | 2019/9/10 |
| 评估测试 | 1个工作日 | 2019/9/11 | 2019/9/11 |

## 输出文档

1. 测试计划
2. 功能测试用例
3. 性能测试方案
4. 性能测试数据
5. bug数据
6. 测试报告

## 测试用例模板

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目名称** |  | | **模块名称** |  | |
| **功能名称** |  | | **开发人员** |  | |
| **测试编号** |  | **测试类型** | 功能测试 | **测试方法** | 手工、黑盒 |
| **测试人员** |  | **测试日期** |  | | |
| **功能描述** |  | | | | |
| **操作步骤** | **预期结果** | **实际结果** | **结果比较说明** | **测试结果截图** | |
|  |  |  |  |  | |

## 测试功能点矩阵

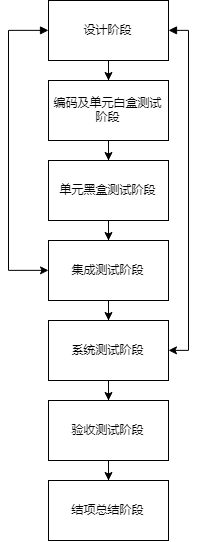
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 功能模块 | 功能内容 | 快捷键 | 功能  是否执行成功 | 快捷键  是否正确 | 执行结果 | 执行截图 |
| 文本编辑 | 复制 | Ctrl + C |  |  |  |  |
| 粘贴 | Ctrl + V |  |  |  |  |
| 查找 | Ctrl + F |  |  |  |  |
| 替换 | Ctrl + H |  |  |  |  |
| 撤销 | Ctrl + Z |  |  |  |  |
| 重做 | Ctrl + Y |  |  |  |  |
| 在状态栏显示当前行数 |  |  |  |  |  |
| 调整字体大小 | Ctrl + wheelEvent |  |  |  |  |
| 代码编辑 | 关键字识别高亮 |  |  |  |  |  |
| 关键字联想提示、补全 | Tab |  |  |  |  |
| 括号自动匹配高亮 |  |  |  |  |  |
| 括号自动补全 |  |  |  |  |  |
| 变量重命名 | F2 |  |  |  |  |
| 自动缩进 |  |  |  |  |  |
| 多行注释 | Ctrl + Shift + / |  |  |  |  |
| 代码块折叠 |  |  |  |  |  |
| 对已编辑的函数进行高亮显示 |  |  |  |  |  |
| 代码跳转 | Ctrl + mouseClick |  |  |  |  |
| 行格式排版 |  |  |  |  |  |
| 整体格式排版 | F3 |  |  |  |  |
| 注释显示 / 隐藏 | F8 |  |  |  |  |
| 代码运行 | 通过gcc对当前文件编译 | Ctrl + B |  |  |  |  |
| 通过gcc对当前文件运行 | Ctrl + R |  |  |  |  |
| 通过gcc实现多文件编译 | Ctrl + Shift + B |  |  |  |  |
| 通过gcc实现项目运行 | Alt + R |  |  |  |  |
| 在控制台显示编译结果 |  |  |  |  |  |
| 通过gdb对当前文件调试 | F5 |  |  |  |  |
| 文件管理 | 新建文件 | Ctrl + N |  |  |  |  |
| 打开文件 | Ctrl + O |  |  |  |  |
| 打开文件夹 | Ctrl + Shift + O |  |  |  |  |
| 保存文件 | Ctrl + S |  |  |  |  |
| 另存为文件 | Ctrl + Shift + S |  |  |  |  |
| 多文件编辑 |  |  |  |  |  |
| 树形文件资源管理 |  |  |  |  |  |

## 测试性能点矩阵

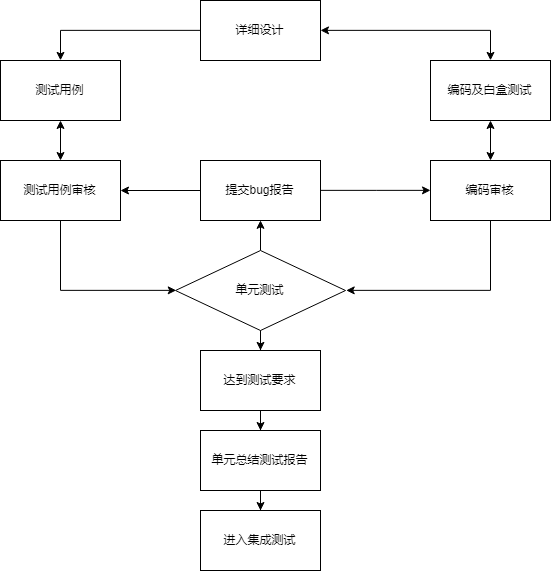
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能模块 | 功能内容 | 响应时间 | 易用性 | 可扩展性 | 健壮性 |
| 文本编辑 | 复制 |  |  |  |  |
| 粘贴 |  |  |  |  |
| 查找 |  |  |  |  |
| 替换 |  |  |  |  |
| 撤销 |  |  |  |  |
| 重做 |  |  |  |  |
| 在状态栏显示当前行数 |  |  |  |  |
| 调整字体大小 |  |  |  |  |
| 代码编辑 | 关键字识别高亮 |  |  |  |  |
| 关键字联想提示、补全 |  |  |  |  |
| 括号自动匹配高亮 |  |  |  |  |
| 括号自动补全 |  |  |  |  |
| 变量重命名 |  |  |  |  |
| 自动缩进 |  |  |  |  |
| 多行注释 |  |  |  |  |
| 代码块折叠 |  |  |  |  |
| 对已编辑的函数进行高亮显示 |  |  |  |  |
| 代码跳转 |  |  |  |  |
| 对类和结构体成员进行提示 |  |  |  |  |
| 行格式排版 |  |  |  |  |
| 整体格式排版 |  |  |  |  |
| 语法错误提示 |  |  |  |  |
| 注释显示 / 隐藏 |  |  |  |  |
| 代码运行 | 通过gcc对当前文件编译 |  |  |  |  |
| 通过gcc对当前文件运行 |  |  |  |  |
| 通过gcc实现多文件编译 |  |  |  |  |
| 通过gcc实现项目运行 |  |  |  |  |
| 在控制台显示编译结果 |  |  |  |  |
| 通过gdb对当前文件调试 |  |  |  |  |
| 文件管理 | 新建文件 |  |  |  |  |
| 打开文件 |  |  |  |  |
| 打开文件夹 |  |  |  |  |
| 保存文件 |  |  |  |  |
| 另存为文件 |  |  |  |  |
| 多文件编辑 |  |  |  |  |
| 树形文件资源管理 |  |  |  |  |
| tabwight显示窗口 |  |  |  |  |

# 执行测试工作流程

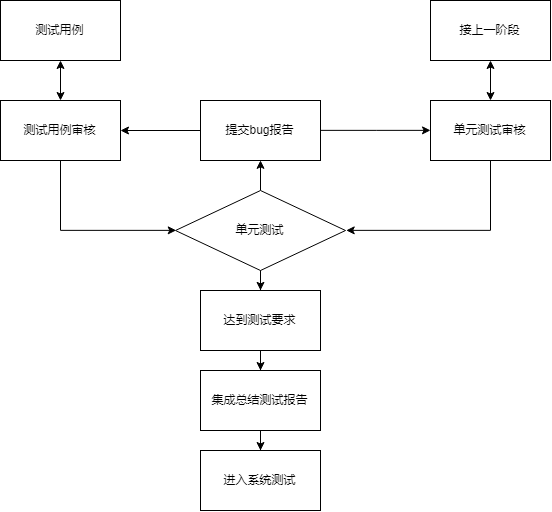
## 测试工作总体流程



## 单元测试工作流程

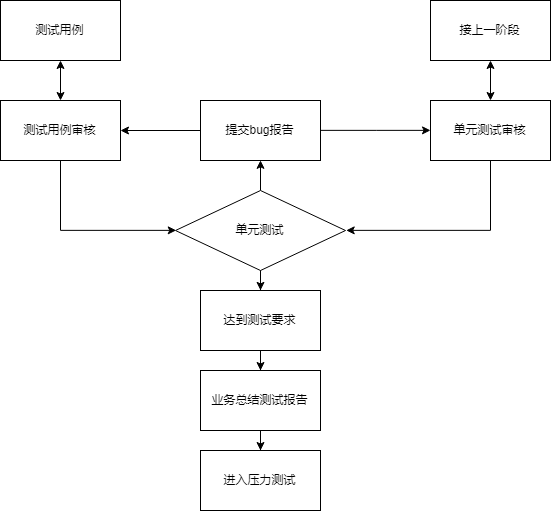


## 集成测试工作流程

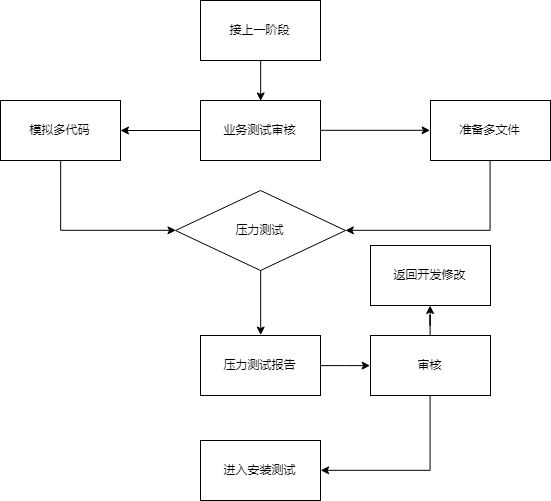


## 系统测试工作流程

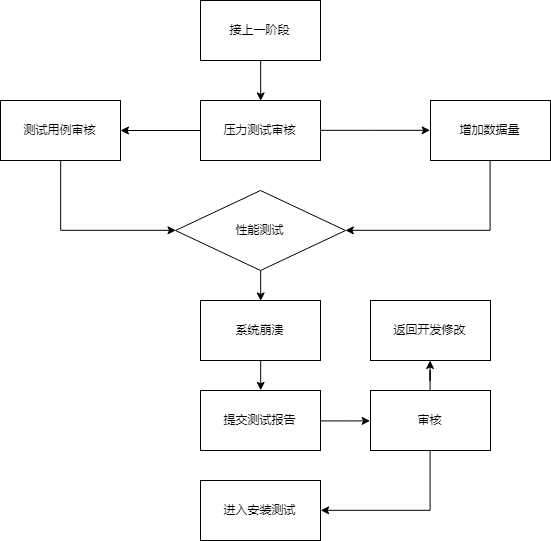
### 业务测试工作流程



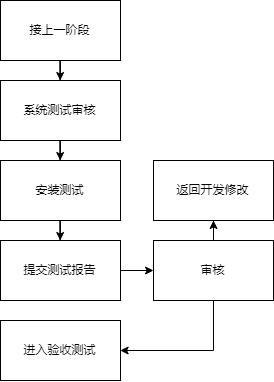
### 压力测试工作流程



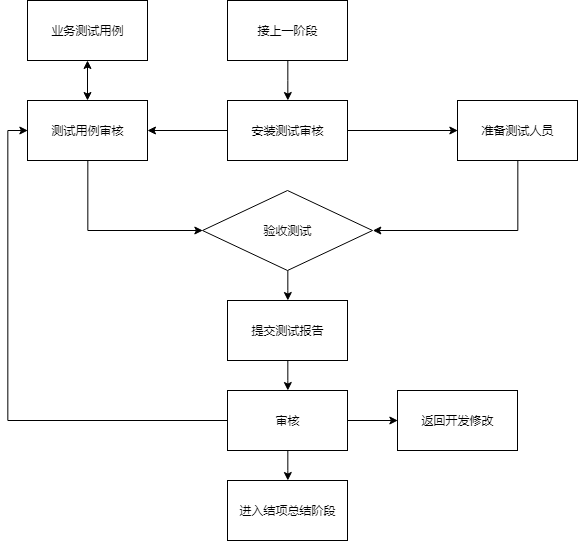
### 性能测试工作流程



### 安装测试工作流程



### 验收测试工作流程



# 发布标准

## 测试完成标准

1. 系统无业务逻辑错误和二级以上的BUG。经确定的所有缺陷都已得到了商定的解决结果。所设计的测试用例已全部重新执行，已知的所有缺陷都已按照商定的方式进行了处理，而且没有发现新的缺陷；
2. 按照集成测试用例完成了整个系统的集成测试>集成版本满足设计定义的各项功能、性能要求；
3. 软件需求分析说明书中定义的所有功能都已经实现，性能指标全部达到性能需求指标；
4. 提交阶段性测试报告，包括功能和性能测试报告；
5. 所有文档齐备完整。

## 产品发布标准

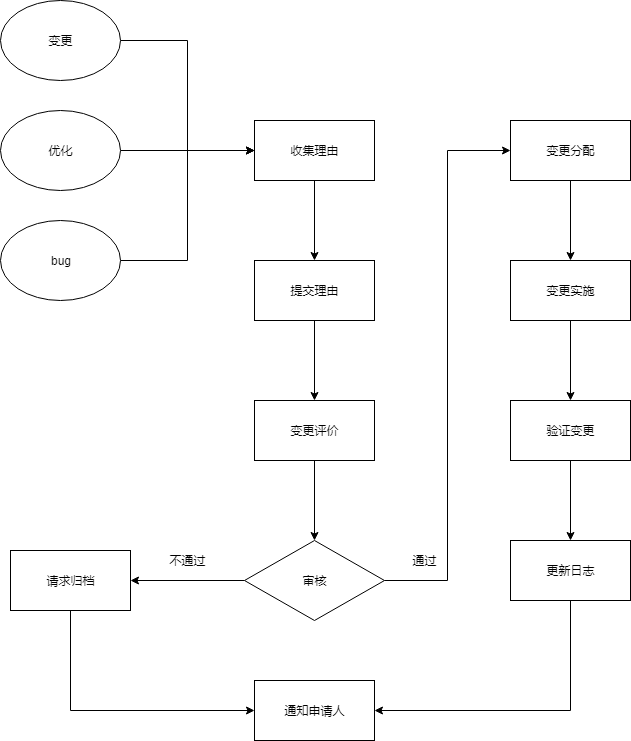
1. 软件产品通过了单元测试、集成测试、业务测试、系统测试、性能测试
2. 测试人员提交文档：测试计划、测试方案、测试用例、测试分析报告
3. 所有测试项必须符合以下标准
4. 致命错误：无
5. 功能错误：无
6. 功能缺陷：项目经理、技术经理、测试负责人审核通过
7. 界面缺陷：项目经理、技术经理、测试负责人审核通过
8. 建议：项目经理、技术经理、测试负责人审核通过
9. 以上几项其中之一不满足要求，视为不合格

# 风险评估

## 变更管理

变更管理包括需求变更、测试案例、测试报告、测试日志等变更。

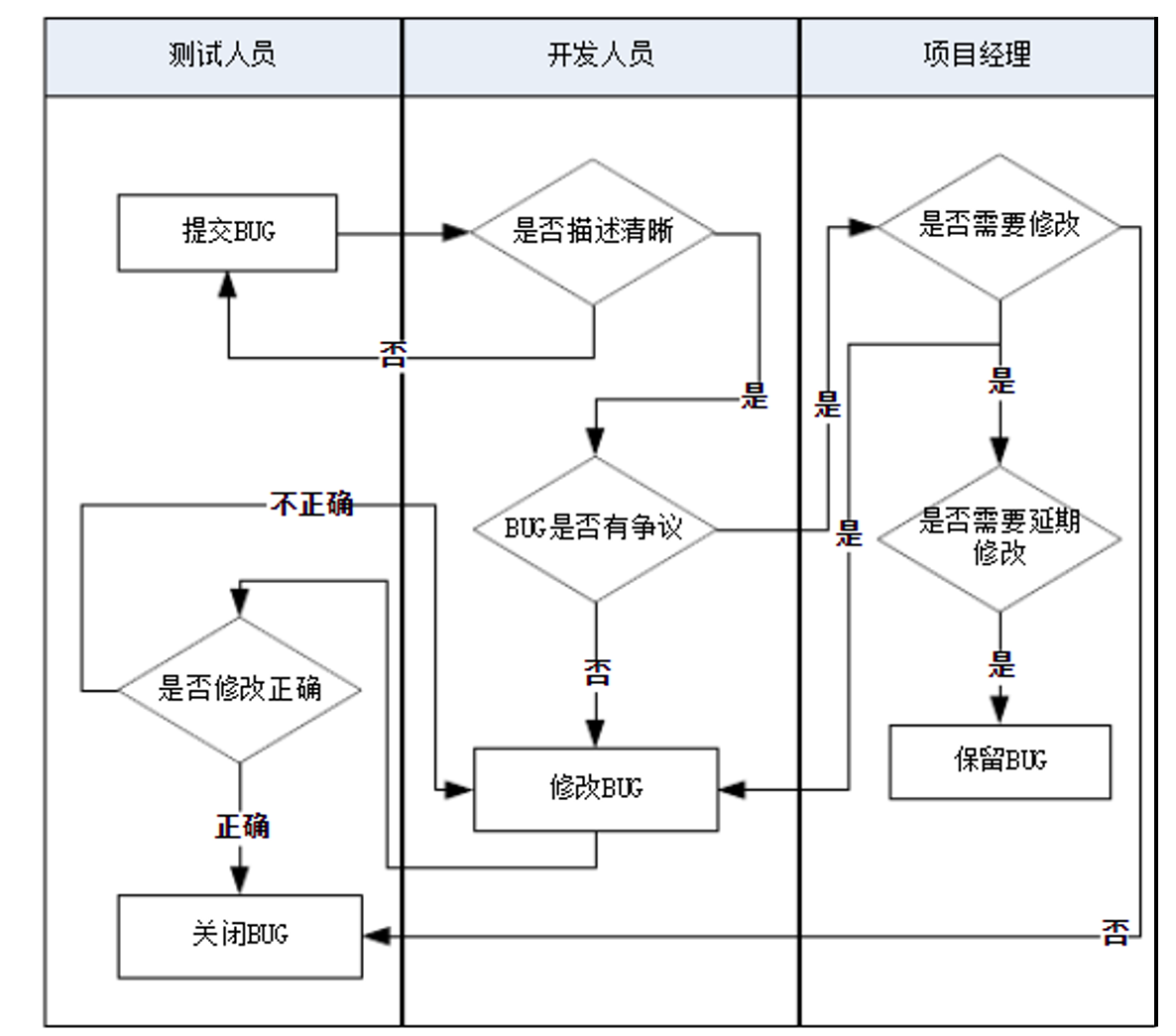
要求规范各类文档的样式和变更说明规范，所有正式的关键性的变更都需要进行变更申请，变更流程如下：



## 缺陷级别

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 缺陷级别 | 缺陷性质 | 系统中对应的错误分类 | 缺陷描述 | 处理方案 |
| 一级 | 致命 | 系统崩渍 系统死锁 | 导致对被描述的主要对象的理解错误、不可行、不可运转、对业务和整个系统造成重大损失或损害； 对使用、维护或保管人员有危险或不安全，以及对产品的基本功能有致命影响的缺陷 | 立即上报项目经理， 项目经理联系软件工程师等技术人员进行修复 |
| 二级 | 严重 | 严重错误 | 对被描述的部分对象的理解或实现错误，部分的模块或系统不可行或不能运转或部分模块和系统缺失，对整个系统有重大影响或可能造成部分的损失或损害；严重影响使用安全 | 一旦发生这样的缺陷，必须进行详细的记录，并且立刻上报给项目组长和项目经理，项目经理立马联系系统架构师及相关人员做修复 |
| 三级 | 一般 | 次要错误 布局不合理 文字错误 | 系统中部分单元模块或单个功能描述和实现有错误、有偏差、不一致或有缺失，不影响模块的正常运行，或有影响，但可以有替代的办法或避免办法 | 综合记录之后立马上报给开发人员进行修改 |
| 四级 | 提示 | 微不足道 | 基本不影响系统的运行和功能的实现。但是与标准、规范和定义不一致 | 及时记录在测试报告中，并且表明问题的描述和截图等信息，与其他错误情况一起提交 |
| 五级 | 建议 | 新特性 | 不在定义、标准、范围的定义和约束之内，但是从提出者来看是需要完善的建议 | 记录在报告中 |

## 缺陷管理流程



# 测试用例

## 基础编辑

1. #include<stdio.h>
2. **int** func(**int** a, **int** b) ;
3. **int** main{
4. **int** a;
5. **int** b;
7. a=10;
8. a=a+b ;
10. func(a,b);
11. }
13. **int** func ( **int** a , **int** b ) {
14. **return** a+b;
15. }

**操作**：逐行输入代码

**预期结果**：输入过程中实时生成代码补全提示，按Tab可以完成补全。每行输入完成按下回车键后改行自动完成排版。

**操作**：按住Ctrl键，同时滚动鼠标滚轮。

**预期结果**：编辑区字体大小随之调整。

**操作**：选择若干行文本，按Ctrl + Shift + /。

**预期结果**：选中行被设为注释。

**操作**：点击变量名，按F2，输入替换文本。

**预期结果**：变量名被成功替换。

**操作**：按Ctrl + F，输入查找文本

**预期结果**：提示用户查找到的信息

**操作**：按Ctrl + H，输入查找文本和替换文本。

**预期结果**：提示用户查找到的信息，完成替换

## 编译运行

### 测试样例1

1. **int** main(){
2. printf("hello world!\n");
3. }

**操作**：输入代码，按Ctrl + B编译。

**预期结果**：提示用户保存。编译提示区输出错误提示。

### 测试样例2

1. #include<stdio.h>
2. **int** func(**int** a, **int** b) ;
3. **int** main{
4. **int** a;
5. **int** b;
7. a=10;
8. a=a+b ;
10. func(a,b);
11. }
13. **int** func ( **int** a , **int** b ) {
14. **return** a+b;
15. }

**操作**：输入代码，按Ctrl + R运行。

**预期结果**：提示用户保存。编译提示区输出未编译提示。

**操作**：按Ctrl + B编译。

**预期结果**：编译提示区输出成功提示。

**操作**：按Ctrl + R运行

**预期结果**：程序正常编译运行。

## 多文件编译运行

### 测试样例1

#### main.cpp

1. #include <iostream>
2. #include <stdio.h>
3. #include "func.h"
4. **using** **namespace** std;
6. **int** main(){
7. **int** j = fact(5);
8. cout << "5! is " << j << endl;
9. system("pause");
10. **return** 0;
11. }

#### func.h

1. **int** mabs(**int** n){
2. **return** (n > 0) ? n : -n;
3. }

**操作**：输入代码，按Ctrl + Shift + B进行多文件编译

**预期结果**：提示用户保存。编译提示区输出错误提示。

### 测试样例2

#### main.cpp

1. #include <iostream>
2. #include <stdio.h>
3. #include "func.h"
4. **using** **namespace** std;

7. **int** main(){
8. **int** j = fact(5);
9. cout << "5! is " << j << endl;
10. **for** (**int** i = 1; i <= 5; ++i)
11. cout << static\_val() << " ";
12. cout << endl;
13. cout << "mabs(-8) is " << mabs(-8) << endl;
14. system("pause");
15. **return** 0;
16. }

#### func.h

1. **int** fact(**int** n);
2. **int** static\_val();
3. **int** mabs(**int**);

#### func.cpp

1. #include "func.h"
3. **int** fact(**int** n){
4. **int** ret = 1;
5. **while** (n > 1)
6. ret \*= n--;
7. **return** ret;
8. }
10. **int** static\_val(){
11. **static** **int** count = 1;
12. **return** ++count;
14. }
16. **int** mabs(**int** n){
17. **return** (n > 0) ? n : -n;
18. }

**操作**：输入代码，按Ctrl + Shift + B进行多文件编译。

**预期结果**：提示用户保存。编译提示区输出成功提示。

**操作**：按Alt + R运行。

**预期结果**：程序可以正常运行。

## 调试

### 测试用例1

1. #include <stdio.h>
2. #include <stdlib.h>
3. **struct** node {
4. **int** a;
5. **int** b;
6. **char** c;
7. };
9. **int** main() {
10. **struct** node\* aaa;
11. aaa->a = 1;
12. printf("%d",aaa->a);
13. **return** 0;
14. }

**操作**：输入代码，按F5开始调试。

**预期结果**：提示用户保存。开始编译。在调试过程中输出错误提示，并正常退出调试。

### 测试用例2

1. #include <stdio.h>
2. **int** main()
3. {
4. **int** n =5;
5. **if** (n == 0)
6. ;
7. **for**(**int** i = 0; i < n; i++)
8. printf("\*");
9. printf("\n");
10. **return** 0;
11. }

**操作**：输入代码，按F5开始调试。

**预期结果**：提示用户保存。开始编译。

**操作**：设置断点，输入待查看变量，开始调试。

**预期结果**：在调试过程中提示运行位置，输出变量值，调试结束后正常退出调试。

## 新需求——注释隐藏折叠

1. //this is an annotation
3. //this is an annotation
4. //this is another annotation
5. **this** is NOT an annotation;
6. /\*this is another annotation, too\*/
7. **this** is NOT an annotation, either;//but this one is
8. /\*
9. \* this is an annotation， too
10. \*/
11. //this is also an annotation
12. **this** is a normal text;/\*but this is annotation\*/**this** is also annotate;

**操作**：输入代码，按F8隐藏所有注释。

**预期结果**：所有注释均被隐藏，仅剩正文文本。

**操作**：按F8显示所有注释。

**预期结果**：还原为初始状态。