Dash Duck 软件工程化实践说明

1. 自动化实践

1.1 构建自动化

```yaml
# 示例 GitHub Actions 配置
name: CI/CD Pipeline

on:
  push:
    branches: [ master, develop ]
  pull_request:
    branches: [ master ]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - uses: subosito/flutter-action@v2
    - run: flutter pub get
    - run: flutter test
    - run: flutter analyze
```

1.2 测试自动化

- **单元测试**: 覆盖核心业务逻辑
- **组件测试**: UI组件功能验证
- **集成测试**: 端到端流程测试

1.3 部署自动化

- **持续集成**: 代码提交触发自动构建
- **持续部署**: 测试通过后自动发布
- **版本管理**: 语义化版本控制

2. 协作化实践

2.1 版本控制

- **Git工作流**: GitFlow分支模型
- **提交规范**: 约定式提交(Conventional Commits)

- **代码审查**: Pull Request必须审查

2.2 开发规范

```
// 代码风格示例
class SchedulePage extends StatefulWidget {
  const SchedulePage({super.key});

  @override
  State<SchedulePage> createState() => _SchedulePageState();
}
```

3. 质量保证

3.1 代码质量

- **静态分析**: flutter analyze
- **代码格式化**: dart format
- **Lint规则**: analysis_options.yaml

3.2 文档质量

- **API文档**: 自动生成文档
- **用户文档**: README.md维护
- **开发文档**: 架构和设计文档

4. 创新实践

4.1 智能化开发

- **热重载**: 提高开发效率
- **代码生成**: 自动生成样板代码
- **智能补全**: IDE集成开发环境

4.2 现代化工具链

- **包管理**: pub包管理器
- **依赖管理**: pubspec.yaml
- **环境管理**: Flutter SDK版本管理