Developer Collaboration Workflow Sequence

Based on my review of the user guide, here are the specific workflows for developer collaboration using vim as the IDE:

Developer Collaboration Workflow Sequence

□□ Developer 1 (Initial Setup)

```
# 1. Create new project and set up repository
mkdir research-project
cd research-project

# 2. Initialize complete research compendium
zzrrtools --dotfiles ~/dotfiles

# 3. Set up version control and push to GitHub
git init
git add .
git commit -m "Initial zzrrtools setup"
git remote add origin https://github.com/[TEAM]/project.git
git push -u origin main

# 4. Start development work in containerized vim environment
make docker-zsh                # → Enhanced zsh shell with personal dotfiles

# 5. Add packages and do initial analysis
# (In zsh container with vim IDE)
R                              # Start R session
# install.packages("tidyverse")
# install.packages("lme4")
# renv::snapshot()
# quit()                       # Exit R

# 6. Create initial analysis scripts using vim
vim R/analysis_functions.R     # Create package functions
# Write R functions with vim + plugins

vim scripts/01_data_import.R   # Create analysis scripts
# Write data import code

vim analysis/paper/paper.Rmd   # Start research paper
# Write analysis and methods in R Markdown

# 7. Quality assurance and commit
```

```
exit                              # Exit container
make docker-check-renv-fix        # Validate dependencies
make docker-test                  # Run package tests
make docker-render                # Test paper rendering

# 8. Commit changes with CI/CD trigger
git add .
git commit -m "Add initial analysis and dependencies"
git push                          # → Triggers GitHub Actions validation
```

👥 Developer 2 (Joining Project)

```
# 1. Clone existing project
git clone https://github.com/[TEAM]/project.git
cd project

# 2. Set up environment (structure already exists)
make docker-build                 # Build container with existing dependencies

# 3. Start development immediately in vim environment
make docker-zsh                   # → Consistent zsh environment with Dev 1

# 4. Sync with latest packages and add new work
# (In zsh container with vim)
R                                 # Start R session
# renv::restore()                 # Get Dev 1's packages
# install.packages("ggplot2")     # Add new package
# renv::snapshot()                # Update environment
# quit()                          # Exit R

# 5. Create visualization functions using vim
vim R/plotting_functions.R        # Add plotting utilities
# Write ggplot2 wrapper functions

vim scripts/02_visualization.R  # Create visualization script
# Write code to generate analysis plots

# 6. Test functions interactively
R                                 # Start R for testing
# devtools::load_all()            # Load package functions
# source("scripts/02_visualization.R")  # Test new code
# quit()

# 7. Quality assurance workflow
exit                            # Exit container
```

```
make docker-check-renv-fix     # Update DESCRIPTION with new packages
make docker-test               # Ensure tests still pass

# 8. Commit with automated validation
git add .
git commit -m "Add visualization analysis with ggplot2"
git push                       # → GitHub Actions validates changes
```

□□ Developer 1 (Continuing Work)

```
# 1. Sync with Developer 2's changes
git pull                        # Get latest code and renv.lock updates

# 2. Rebuild environment with new dependencies
make docker-build              # Rebuild container with Dev 2's packages

# 3. Validate environment consistency
make docker-check-renv-fix     # Ensure all dependencies are properly tracked

# 4. Continue development with updated environment
make docker-zsh                # → Environment now includes Dev 2's packages

# 5. Add more analysis work using vim
# (In zsh container with vim)
R                              # Start R session
# renv::restore()              # Ensure all packages from Dev 2 are available
# devtools::load_all()         # Load updated package with new functions
# quit()

# 6. Create advanced analysis using vim
vim R/modeling_functions.R     # Add statistical modeling functions
# Write multilevel model functions

vim scripts/03_advanced_models.R  # Create modeling script
# Write analysis using both Dev 1 and Dev 2's functions

# 7. Test integration of both developers' work
R                              # Interactive testing
# devtools::load_all()         # Load all functions
# source("scripts/01_data_import.R")    # Dev 1's work
# source("scripts/02_visualization.R") # Dev 2's work
# source("scripts/03_advanced_models.R") # New integration
# quit()

# 8. Update research paper with new analysis
```

```
vim analysis/paper/paper.Rmd  # Update manuscript
# Add new results and figures

# 9. Enhanced collaboration workflow
exit                          # Exit container

# 10. Use enhanced GitHub templates for pull request
git checkout -b feature/advanced-models
git add .
git commit -m "Add multilevel models integrating visualization functions"
git push origin feature/advanced-models

# 11. Create pull request using enhanced template
# GitHub automatically provides:
# - Analysis impact assessment checklist
# - Reproducibility validation
# - Automated CI/CD checks
# - Paper rendering validation
```

□ Key Collaboration Features (rrtools_plus Integration)

Automated Quality Assurance on Every Push:

- □ **R Package Validation**: R CMD check with dependency validation
- □ **Paper Rendering**: Automated PDF generation and artifact upload
- □ **Multi-platform Testing**: Ensures compatibility across environments
- □ **Dependency Sync**: renv validation and DESCRIPTION file updates

Enhanced GitHub Templates:

- **Pull Request Template**: Analysis impact assessment, reproducibility checklist
- **Issue Templates**: Bug reports with environment details, feature requests with research use cases
- **Collaboration Guidelines**: Research-specific workflow standards

Seamless Environment Synchronization:

```
# Any developer can sync at any time:
git pull                       # Get latest changes
make docker-build              # Rebuild with updated dependencies
make docker-zsh                # → Identical vim/zsh environment across team
```

Data Management Collaboration:

```
# Structured data workflow for teams:
data/
├── raw_data/                # Dev 1 adds original datasets
├── derived_data/            # Dev 2 adds processed data
├── metadata/                # Both document data sources
└── validation/              # Automated quality reports
```

Vim IDE Development Environment

Enhanced Vim Setup (via zzrrtools dotfiles)

The containerized environment includes a fully configured vim IDE with:

Vim Plugin Ecosystem:

- **vim-plug**: Plugin manager (automatically installed)
- **R Language Support**: Syntax highlighting and R integration
- **File Navigation**: Project file browser and fuzzy finding
- **Git Integration**: Git status and diff visualization
- **Code Completion**: Intelligent autocomplete for R functions

Essential Vim Workflow Commands:

```
# In container vim session:
vim R/analysis.R               # Open R file
:Explore                       # File browser
:split scripts/data.R          # Split window editing
:vsplit analysis/paper.Rmd     # Vertical split for manuscript

# Vim + R integration:
:terminal                      # Open terminal in vim
R                             # Start R session in terminal
# devtools::load_all()         # Load package functions (in R)
# :q                           # Exit R, back to vim

# Git workflow in vim:
:!git status                   # Check git status
:!git add %                    # Add current file
:!git commit -m "Update analysis"  # Commit changes
```

Productive Development Cycle:

```
# 1. Start development environment
make docker-zsh                # → Enhanced zsh with vim
```

```
# 2. Multi-file development workflow
vim -p R/functions.R scripts/analysis.R analysis/paper/paper.Rmd
# Opens multiple files in tabs

# 3. Interactive R testing
:terminal                      # Open terminal in vim
R                             # Start R
# devtools::load_all()         # Test functions
# source("scripts/analysis.R") # Test scripts
# quit()                       # Exit R

# 4. File navigation and editing
# gt (next tab), gT (previous tab)
# Ctrl+w+w (switch windows)
# :Explore (file browser)

# 5. Quick testing cycle
:!make docker-test           # Run tests from vim
:!make docker-render         # Render paper from vim
```

Vim + R Development Tips:

File Organization in Vim:

```
# Open related files simultaneously:
vim -O R/analysis_functions.R scripts/01_analysis.R    # Side by side
vim -o R/plotting.R analysis/figures/                  # Horizontal split
vim -p R/*.R scripts/*.R                               # All R files in tabs
```

Git Integration Workflow:

```
# In vim, check git status frequently:
:!git status                   # See changed files
:!git diff %                   # Diff current file
:!git add %                    # Stage current file
:!git commit -m "Add function"  # Commit from vim

# View git log:
:!git log --oneline -10        # Recent commits
```

R Package Development in Vim:

```
# Typical development cycle:
vim R/new_function.R         # Write function
:!make docker-test           # Test from vim
```

```
vim man/new_function.Rd      # Check documentation
:!make docker-check          # Package validation
```

This workflow ensures **perfect reproducibility** across team members while providing **automated quality assurance** and **professional collaboration tools** integrated from the rrtools_plus enhancement framework, all accessible through a powerful vim-based development environment.