

Introducing smalltalk-dev-plugin

AI-Driven Development Toolkit for Pharo Smalltalk

Masashi Umezawa

<https://github.com/mumez/smalltalk-dev-plugin>

What is smalltalk-dev-plugin?

A comprehensive toolkit that bridges **AI code editors** and the **Pharo Smalltalk** environment.

Using AI coding agents with Smalltalk doesn't work well out of the box:

- **No knowledge of Pharo** — AI doesn't know how to interact with the environment
- **Unknown project structure** — Typical Tonel/package layout is unfamiliar
- **No testing/debugging skills** — Can't run tests or diagnose errors
- **No library access** — Can't browse existing classes or methods
- **Poor coding style** — Doesn't know Smalltalk idioms and conventions

This is a major barrier for developers who want AI assistance in Smalltalk.

Enter smalltalk-dev-plugin

We gave AI the complete skillset for Smalltalk development:

Challenge	Solution
Environment interaction	MCP tools for Pharo communication
Project structure	<code>/st:setup-project</code> with templates
Testing & debugging	<code>/st:test</code> , <code>/st:eval</code> , debugger skill
Library navigation	Finder skills, code search tools
Code quality	Validator, linter, commenter agent

Result: AI becomes a reliable Smalltalk development partner!

"AI editor as the source of truth"

| *Edit Tonel files in the AI editor, import to Pharo, test, and iterate.*

At a Glance

- 9 slash commands
- 4 specialized AI skills
- 27+ MCP tools for Pharo interaction
- Automated code quality checks and documentation generation

Agent	Support Level
Claude Code	Full support (primary)
Cursor	Simplified setup via script
Windsurf	Simplified setup via script
Antigravity	Simplified setup via script

The rest of this presentation focuses on the Claude Code version.

Installation

Installation (1) — Install the Plugin

```
# Add marketplace from GitHub
claude plugin marketplace add mumez/smalltalk-dev-plugin

# Install the plugin
claude plugin install smalltalk-dev
```


Installation (2) — Set Up Pharo Side

9

Install the interop server in Pharo:

```
Metacello new  
  baseline: 'PharoSmalltalkInteropServer';  
  repository: 'github://mumez/PharoSmalltalkInteropServer:master/src';  
  load.
```

Start the server:

```
SisServer current start.
```

That's it — you're ready to go.

Setup scripts are provided for Cursor, Windsurf, and Antigravity:

```
./extra/setup-cursor.sh [target-directory]  
./extra/setup-windsurf.sh [target-directory]  
./extra/setup-antigravity.sh [target-directory]
```

Limitations

- Commands use `st -` prefix instead of `st :`
- Some features may differ depending on the agent
- Pharo-side setup is the same as for Claude Code

Basic Usage

`/st:buddy` is a friendly assistant that routes to the right tool.

Just describe what you want in natural language:

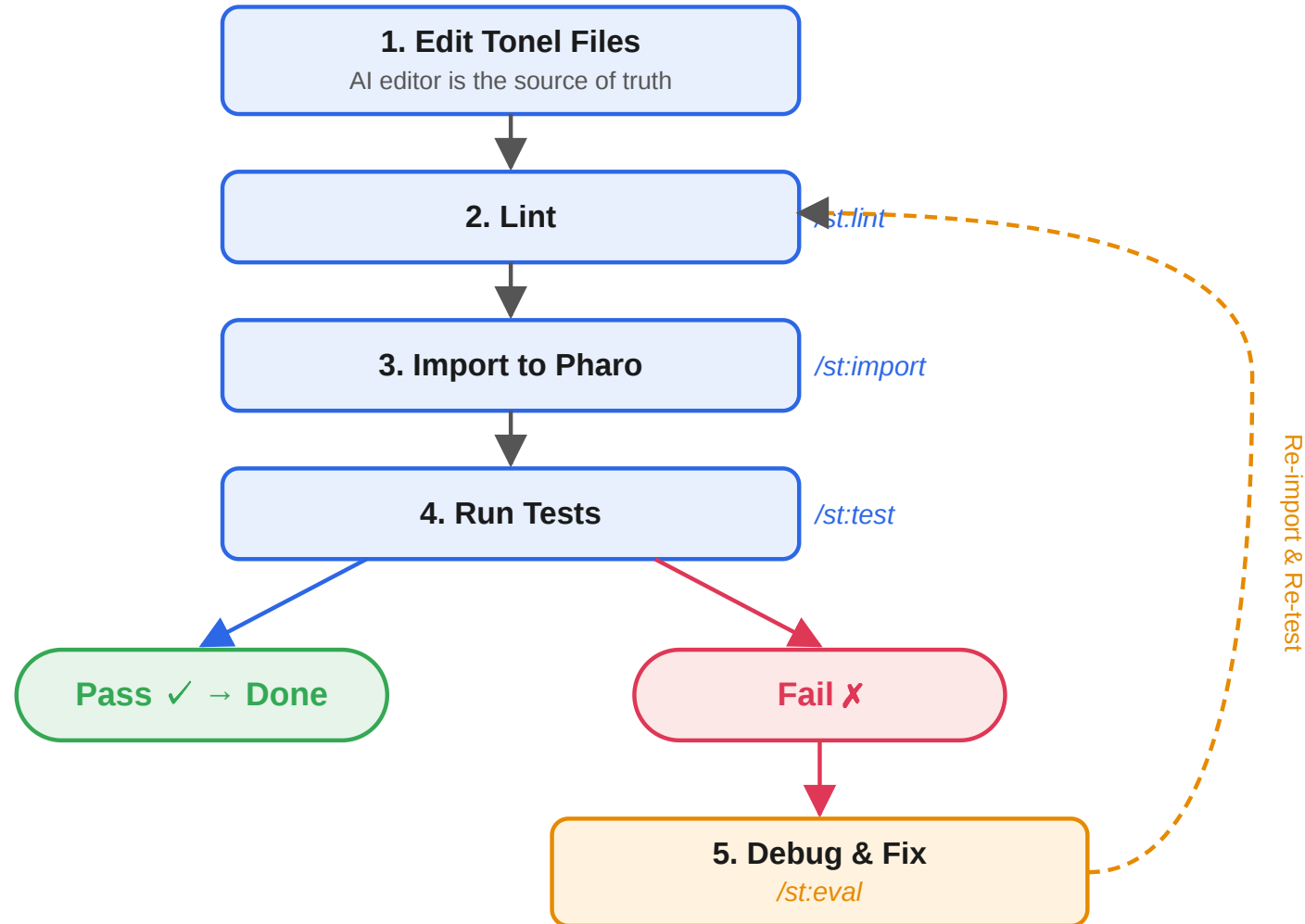
```
/st:buddy I want to create a Person class with name and age
```

The assistant will:

1. Load the appropriate skill automatically
2. Create the Tonel files
3. Guide you through linting, importing, and testing

Your Intent	Loaded Skill
"Create...", "Add..."	smalltalk-developer
"Error...", "Test failed..."	smalltalk-debugger
"How do I use...?"	smalltalk-usage-finder
"Who implements...?"	smalltalk-implementation-finder

You don't need to remember which tool to use — just talk to `/st:buddy`.



Components

smalltalk-developer

The core development skill. Knows Tonel format, package structure, and the full Edit → Lint → Import → Test workflow.

smalltalk-debugger

Systematic debugging specialist. Diagnoses errors, runs incremental code execution with `/st:eval`, and guides you to the fix.

smalltalk-usage-finder

Discovers how classes and methods are used. Finds examples, analyzes usage patterns, and helps you understand unfamiliar APIs.

smalltalk-implementation-finder

Finds method implementations across class hierarchies. Useful for learning coding idioms and assessing refactoring impact.

Command	Purpose
<code>/st:buddy</code>	Friendly assistant (main entry)
<code>/st:init</code>	Initialize development session
<code>/st:setup-project</code>	Create project boilerplate
<code>/st:eval</code>	Execute Smalltalk expressions
<code>/st:import</code>	Import Tonel packages to Pharo
<code>/st:export</code>	Export packages from Pharo
<code>/st:test</code>	Run SUnit tests
<code>/st:lint</code>	Check code quality
<code>/st:validate</code>	Validate Tonel syntax

Automatically suggests **CRC-style class comments** for undocumented classes.

- Triggered after file edits via hooks
- Non-intrusive — suggests, doesn't force
- Analyzes complexity and prioritizes classes that need documentation

Two MCP servers power the integration behind the scenes:

pharo-interop (22 tools)

Pharo communication — eval, import/export, tests, code navigation, etc.

smalltalk-validator (5 tools)

Tonel validation and linting.

Note: You rarely need to interact with these directly.

`/st:buddy` handles orchestration for you.

Live Examples

Example 1 — Starting from Scratch

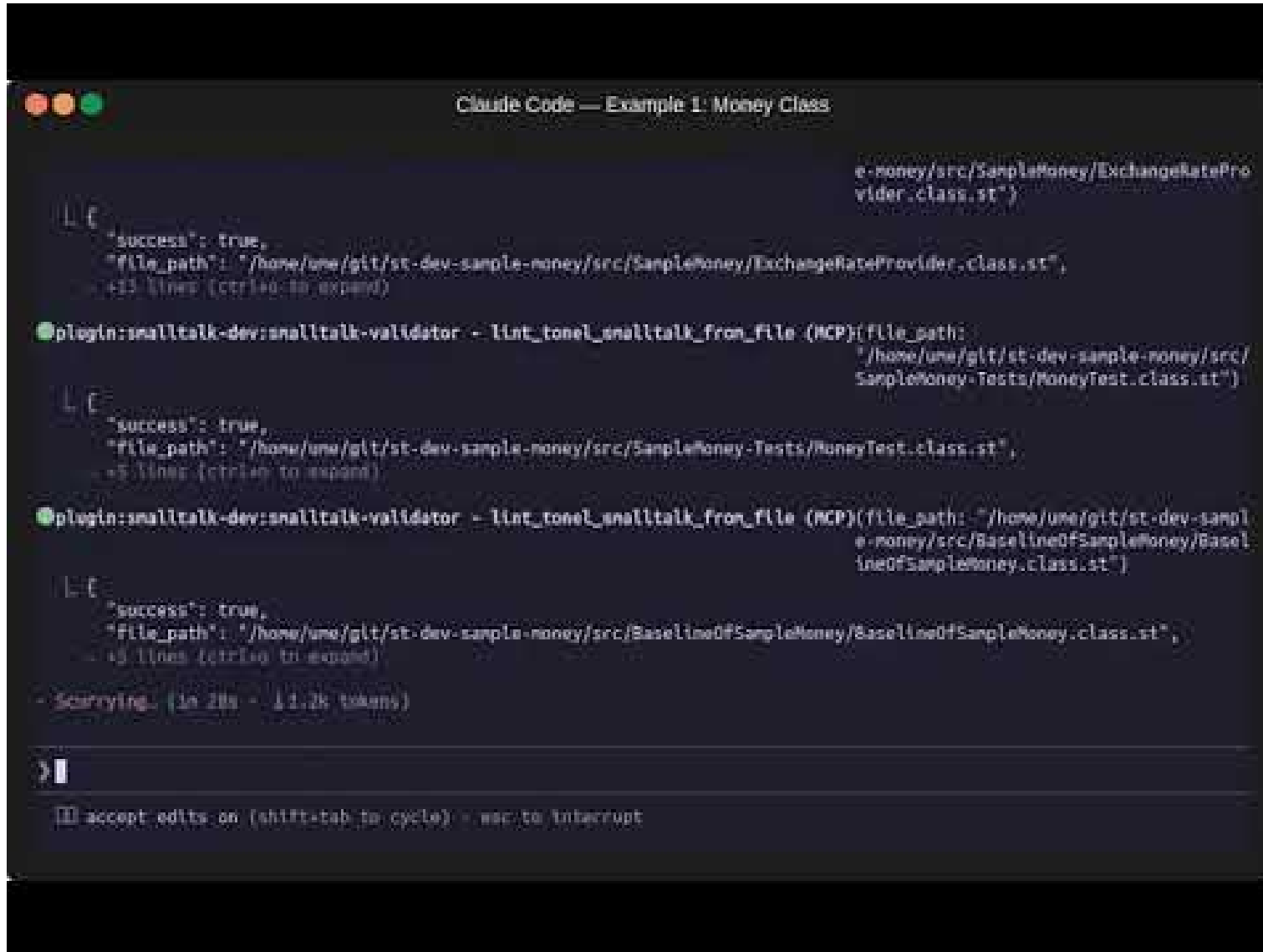
A simple use case: starting from an empty directory as a Smalltalk beginner.

```
/st:buddy I'm a Smalltalk beginner. I want to create a Money class  
that can perform arithmetic operations across different currency units.  
Please help me from the project initialization.
```

Example 1 — What Happens Next

23

- Generated source
- Demo



The screenshot shows the Claude Code interface with the title "Claude Code — Example 1: Money Class". It displays three generated code blocks, each preceded by a green dot icon. The first block is a JSON object with "success": true and a file path. The second block is a function call to "lint_tonel_snalltalk_from_file" with a file path. The third block is another function call to "lint_tonel_snalltalk_from_file" with a file path. At the bottom, there is a status bar indicating "Scarrying... (in 28s - 11.2k tokens)" and a prompt to "accept edits on (shift+tab to cycle) - esc to interrupt".

```
Claude Code — Example 1: Money Class

e-money/src/SampleMoney/ExchangeRateProvider.class.st")

L E
"success": true,
"file_path": "/home/una/git/st-dev-sample-money/src/SampleMoney/ExchangeRateProvider.class.st",
+11 lines (ctrl+o to expand)

plugin:snalltalk-dev:snalltalk-validator - lint_tonel_snalltalk_from_file (MCP)(file_path:
"/home/una/git/st-dev-sample-money/src/
SampleMoney-Tests/MoneyTest.class.st")

L E
"success": true,
"file_path": "/home/una/git/st-dev-sample-money/src/SampleMoney-Tests/MoneyTest.class.st",
+5 lines (ctrl+o to expand)

plugin:snalltalk-dev:snalltalk-validator - lint_tonel_snalltalk_from_file (MCP)(file_path: "/home/una/git/st-dev-sampl
e-money/src/BaselineOfSampleMoney/Basel
ineOfSampleMoney.class.st")

L E
"success": true,
"file_path": "/home/una/git/st-dev-sample-money/src/BaselineOfSampleMoney/BaselineOfSampleMoney.class.st",
+9 lines (ctrl+o to expand)

- Scarrying... (in 28s - 11.2k tokens)

> |

[ ] accept edits on (shift+tab to cycle) - esc to interrupt
```

Example 2 — Graph Algorithms

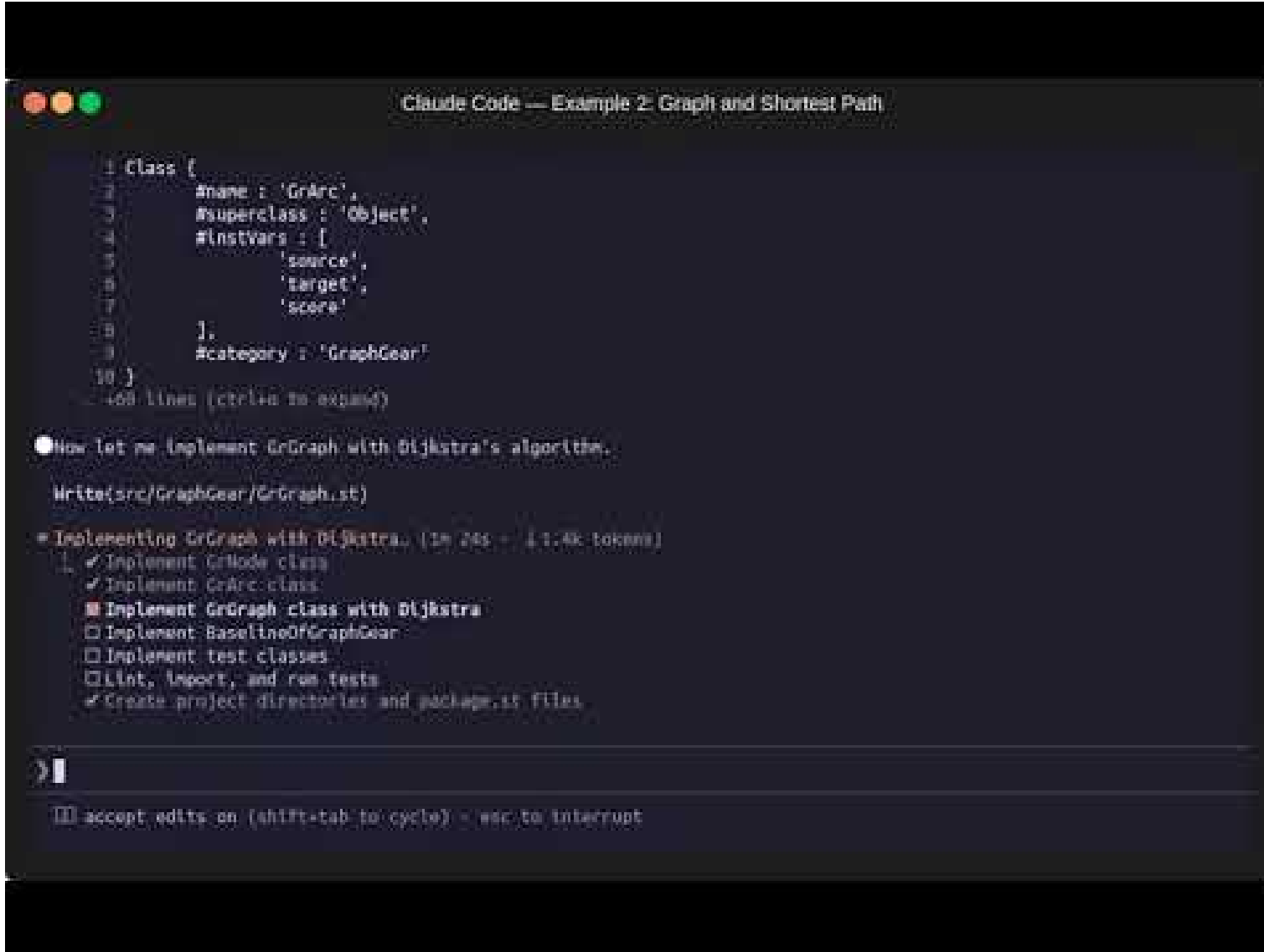
A more complex use case for someone familiar with Smalltalk.

```
/st:buddy I want to create GrNode and GrArc to represent directed graphs  
and solve shortest path problems. Nodes have a name, arcs have a score.  
Let's start this as a GraphGear project.
```


Example 2 — What Happens Next

25

- [Generated source](#)
- [Demo](#)



```
Claude Code — Example 2: Graph and Shortest Path

1 class [
2     #name : 'GrArc',
3     #superclass : 'Object',
4     #instance_vars : [
5         'source',
6         'target',
7         'score'
8     ],
9     #category : 'GraphGear'
10 ]
11 }
12 468 lines (ctrl+u to expand)

Now let me implement GrGraph with Dijkstra's algorithm.

Write(src/GraphGear/GrGraph.st)

= Implementing GrGraph with Dijkstra. (1m 24s - 11.4k tokens)
1. ✓ Implement GrNode class
2. ✓ Implement GrArc class
3. ■ Implement GrGraph class with Dijkstra
4. □ Implement BaselineGrGraphGear
5. □ Implement test classes
6. □ Lint, import, and run tests
7. ✓ Create project directories and package.st files

> |

accept edits on (shift+tab to cycle) - esc to interrupt
```

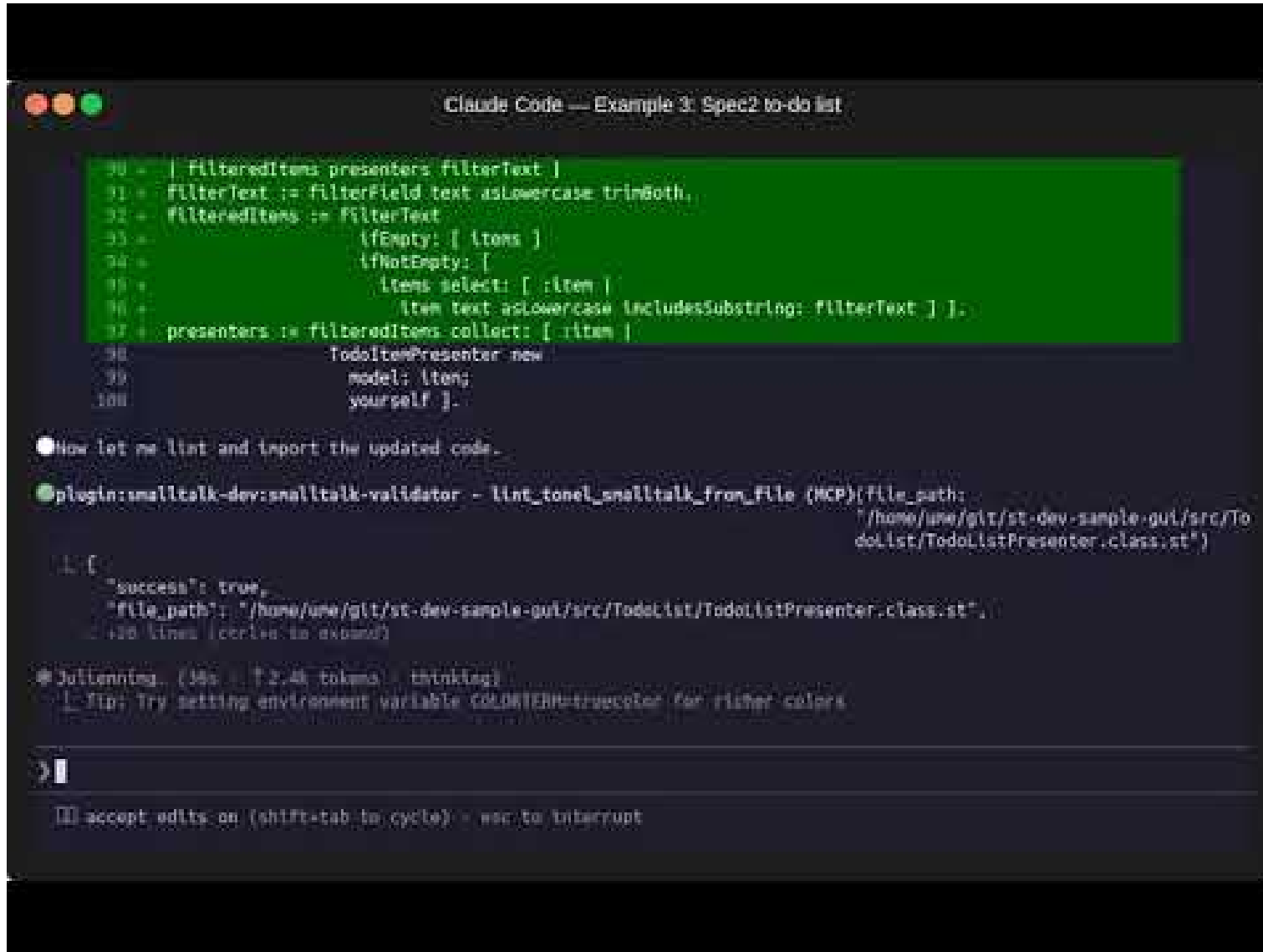
Building a GUI application with Spec2.

```
/st:buddy I want to make a simple to-do list using the Spec2 framework.  
Include a checkbox for each item, plus an input field  
with Add/Remove buttons at the bottom.  
Only checked items can be removed. Start developing.
```

Example 3 — What Happens Next

27

- [Generated source](#)
- [Demo](#)



The screenshot shows a Claude Code window titled "Claude Code — Example 3: Spec2 to-do list". It displays a Smalltalk code snippet with line numbers 90 to 100. The code defines a filter for a to-do list and creates a presenter. Below the code, there is a JSON response from a plugin, a status message from Julianne, and a tip about setting an environment variable.

```
90 + | filteredItems presenters filterText |
91 + filterText := filterField text asLowercase trimBoth.
92 + filteredItems := filterText
93 +     ifEmpty: [ items ]
94 +     ifNotEmpty: [
95 +         items select: [ :item |
96 +             item text asLowercase includesSubstring: filterText ] ].
97 + presenters := filteredItems collect: [ :item |
98 +     TodoItemPresenter new
99 +         model: item;
100 +         yourself ].
```

Now let me list and import the updated code.

```
{
  "success": true,
  "file_path": "/home/une/git/st-dev-sample-gui/src/TodoList/TodoListPresenter.class.st",
  "lines": 130
}
```

Julianne. (38s • 12.4k tokens • thinking)

Tip: Try setting environment variable COLORTERM=truecolor for richer colors

accept edits on (shift+tab to cycle) • esc to interrupt

smalltalk-dev-plugin makes Pharo Smalltalk development with AI assistants practical and productive.

Just type `/st:buddy` and start building.

Feedback and contributions are welcome!

<https://github.com/mumez/smalltalk-dev-plugin>