Package-Aware Approach for Repository-Level Code Completion in Pharo

Omar Abedelkader[1] Stéphane Ducasse[1] Oleksandr Zaitsev[2] Romain Robbes[3]
Guillermo Polito[1]

[1]Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[2]CIRAD, UMR SENS, F-34000 Montpellier, France
[3]CNRS, University of Bordeaux, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France

1st July 2025

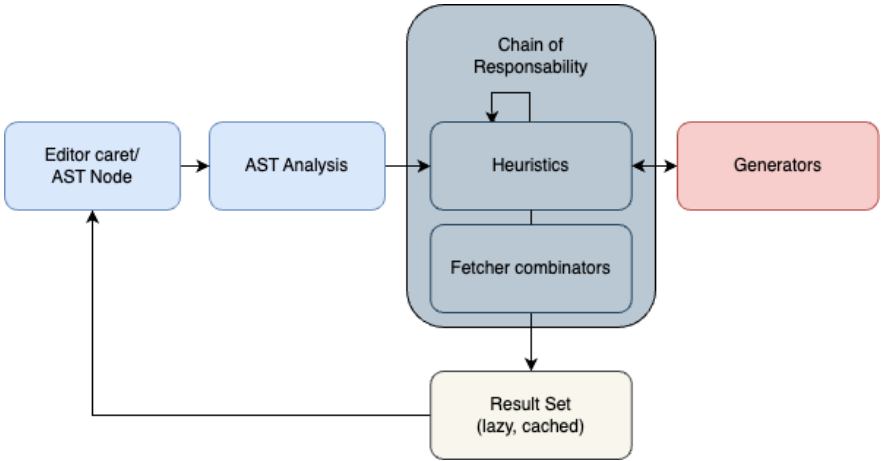
What is Code Completion?

- Assists developers by suggesting possible code elements as they type.
- It helps speed up development, reduces syntax errors, and improves code accuracy.
- Advanced code completion systems offer context-aware suggestions.
- This feature is widely integrated into IDEs and text editors to enhance coding efficiency and developer experience.

What is Complishon ?

- Complishon is an advanced semantic code completion engine designed to enhance coding efficiency.
- Thanks to Guille!

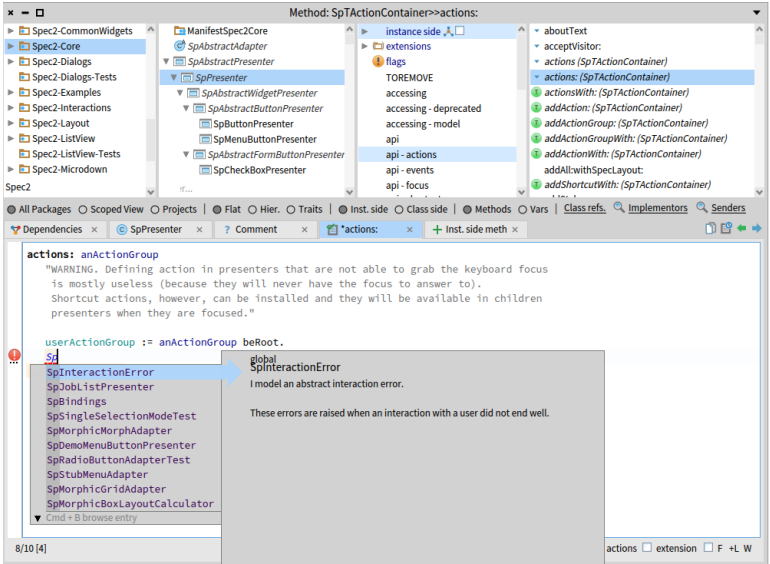
How Complishon works?



Why This Matters ?

- Without context-awareness, developers waste time scanning long, irrelevant suggestion lists.
- In large codebases, global suggestions are overwhelming.
- Package-aware completion provides more relevant and more intuitive suggestions.

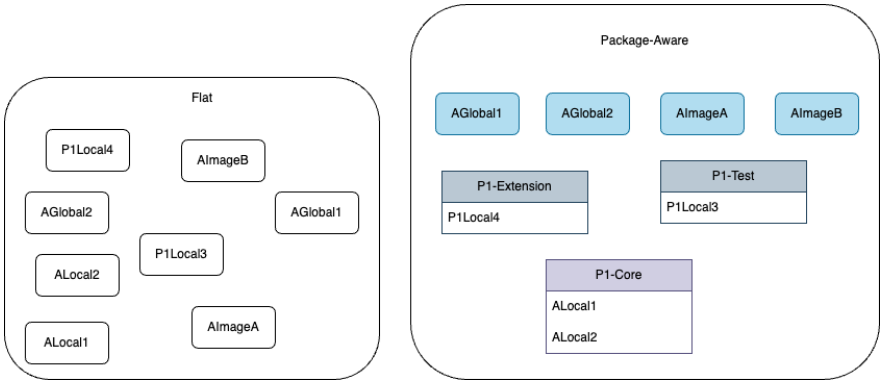
Example?


Method: SpTActionContainer>>actions:
Spec2-CommonWidgets
Spec2-Core
Spec2-Dialogs
Spec2-Dialogs-Tests
Spec2-Examples
Spec2-Interactions
Spec2-Layout
Spec2-ListView
Spec2-ListView-Tests
Spec2-Microdown
Spec2
ManifestSpec2Core
SpAbstractAdapter
SpAbstractPresenter
SpPresenter
SpAbstractWidgetPresenter
SpAbstractButtonPresenter
SpButtonPresenter
SpMenuButtonPresenter
SpAbstractFormButtonPresenter
SpCheckBoxPresenter
instance side
extensions
flags
TOREMOVE
accessing
accessing - deprecated
accessing - model
api
api - actions
api - events
api - focus
aboutText
acceptVisitor:
actions (SpTActionContainer)
actions: (SpTActionContainer)
actionsWith: (SpTActionContainer)
addAction: (SpTActionContainer)
addActionGroup: (SpTActionContainer)
addActionGroupWith: (SpTActionContainer)
addActionWith: (SpTActionContainer)
addAll:withSpecLayout:
addShortcutWith: (SpTActionContainer)
All Packages
Scoped View
Projects
Flat
Hier.
Traits
Inst. side
Class side
Methods
Vars
Class refs.
Implementors
Senders
Dependencies
SpPresenter
Comment
*actions:
Inst. side meth
actions: anActionGroup
"WARNING. Defining action in presenters that are not able to grab the keyboard focus
is mostly useless (because they will never have the focus to answer to).
Shortcut actions, however, can be installed and they will be available in children
presenters when they are focused."
userActionGroup := anActionGroup beRoot.
Sp
SpInteractionError
SpJobListPresenter
SpBindings
SpSingleSelectionModeTest
SpMorphicMorphAdapter
SpDemoMenuButtonPresenter
SpRadioButtonAdapterTest
SpStubMenuAdapter
SpMorphicGridAdapter
SpMorphicBoxLayoutCalculator
Cmd + B browse entry
global
SpInteractionError
I model an abstract interaction error.
These errors are raised when an interaction with a user did not end well.
8/10 [4]
actions
extension
F +L W

The Problem

- **Current Limitation:** Complishon treats the global namespace as a flat list.
- **Result:** Developers are flooded with irrelevant suggestions in large repositories.
- **Why It's a Problem:**
  - Developers typically work within their package.
  - Suggestions from unrelated packages slow them down.

What is Package Awareness?



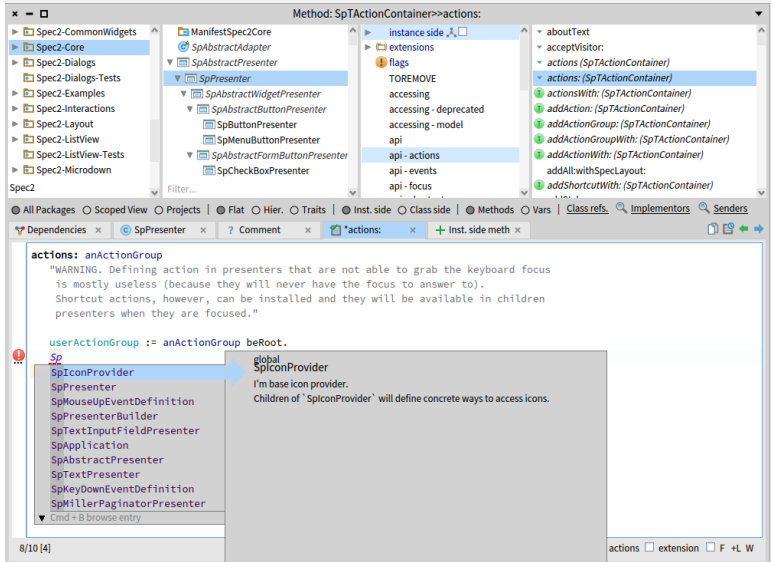
Why Package Awareness?

- Developers typically work within a local scope.
- Nearby classes are more relevant than distant ones.
- Developers want to see their variables from their packages more often than from other packages.

Core Idea

- Developers typically focus on their immediate working context: their current package and nearby packages.
- **Our strategy: Prioritize completion suggestions based on proximity.**
  - **First:** Suggestions from the *same package*.
  - **Second:** Suggestions from *related or nearby packages*.
  - **Last:** Suggestions from the *global namespace*.

After implementing Package Awareness



Benchmark Methodology

- Prefix truncation from 2 to 8 characters.
- Completion invoked for each prefix.
- Metrics: Accuracy, MRR, NDCG.

Mean Reciprocal Rank (MRR)

- **Mean Reciprocal Rank (MRR)** measures how quickly the correct suggestion appears in the completion list.
- For each completion:
  - Reciprocal Rank $= \frac{1}{\text{position of the correct suggestion}}$
- **Higher MRR** means developers find correct completions faster.

Formula:

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{rank}_i}$$

Evaluation Protocols

- Measured Mean Reciprocal Rank (MRR) across multiple frameworks: Iceberg, Moose, Roassal, Seaside, Spec.
- Best Improvements:
  - Spec: +7.59% MRR
  - Iceberg: +6.09% MRR
  - Seaside: +4.72% MRR

More results

Framework	Package Type	Avg. Δ MRR
Iceberg	Overall	6.09
Moose	Overall	1.05
Moose	Test	0.31
Moose	Non-test	1.19
Roassal	Overall	0.90
Roassal	Test	-2.62
Roassal	Non-test	2.14
Seaside	Overall	4.72
Seaside	Test	3.55
Seaside	Non-test	5.33
Spec	Overall	7.59
Spec	Test	2.64
Spec	Non-test	10.31

Table: Summary of Average Δ MRR Across Frameworks and Package Types

Limitations

- **Framework Sensitivity:** Package awareness may not always be beneficial (e.g., test packages).
- **Granularity Dependency:** Minimal improvements in cases like Moose (Test) suggest that if packages are too broad or poorly defined.
- **Test vs. Non-Test** Sometimes reference external classes more often than local ones.
- **Naming Patterns:** Heuristics may occasionally misprioritize.

Future Directions

- Multi-dimensional fetcher dependency
- History Fetcher Approach
- GUI Fetcher Approach
- LLM Approach

Summary

- Package-aware completion improves relevance in Pharo.
- Measured significant MRR improvements in key frameworks.
- Dependency analysis, history tracking, LLM integration.

Support Slide

The Complishon Engine

- Heuristics: Analyze AST nodes using the Chain of Responsibility to route completion logic.
- Lazy Fetchers: Use combinators and decorators to generate and filter suggestions efficiently.
- Result Set: A lazy, cached collector for relevant completions, improving memory and runtime performance.

Code Completion Levels (SoA)

- Single-line: JetBrains FLCC
- Multi-line: Meta's CodeCompose
- **Class/Repo-level:** ClassEval, GraphCoder, RepoCoder TOOLGEN

Code Completion Types (SoA)

- Heuristics and Rules: Context-aware filters, usage frequency, type hierarchies
- Statistical Models: N-gram and probabilistic approaches: SLAMC, SLANG, DEEP3
- Neural Models: PHOG, usability-focused, open-vocab, GPT-style generation.
- AST-Aware Models: TreeGen , TreeBERT , ReGCC, AST-T5
- **Low-Ressource Languages**: Transfer learning, IR translation, kNM-LM, SPEAC