Contents

Chapter 1

Evaluation

Section **??** described the implementation of the Nickel Language Server addressing the first research question stated in sec. **??**. Proving the viability of the result and answering the second research question demands an evaluation of different factors.

Earlier, the most important metrics of interest were identified as:

Usability What is the real-world value of the language server?
Does it improve the experience of developers using Nickel? NLS offers several features, that are intended to help developers using the language. The evaluation should assess whether developers experience any help due to the use of the server.
Does NLS meet its users' expectations in terms of completeness and correctness and behavior? Being marketed as a Language Server, invokes certain expectations due to previous experience with other languages and language servers. Here, the evaluation should show whether NLS lives up to the expectations of its users.

Performance What are the typical latencies of standard tasks? In this context *latency* refers to the time it takes from issuing an LSP command to return of the reply by the server. The JSON-RPC protocol used by the LSP is synchronous, i.e. requires the server to return results of commands in the order it received them. Since most commands are sent implicitly, a quick processing is imperative to avoid commands queuing up.
Can single performance bottlenecks be identified? Single commands with excessive runtimes can slow down the entire communication resulting in bad user experience. Identified issues can guide the future work on the server.
How does the performance of NLS scale for bigger projects? With increasing project sizes the work required to process files increases as well. The evaluation should allow estimates of the sustained performance in real-world scenarios.

Answering the questions above, this chapter consists of two main sections. The first section sec. 1.1 introduces methods employed for the evaluation. In particular, it details the survey (sec. 1.1.1) which was conducted with the intent to gain qualitative opinions by users, as well as the tracing mechanism (sec. 1.1.2) for factual quantitative insights. Section 1.2 summarises the results of these methods.

1.1 Methods

1.1.1 Qualitative

1.1.2 Quantitative

1.2 Results

1.2.1 Process

1.2.2 Qualitative

As outlined in [#sec:qualitative-study-outline], the qualitative study consists of two parts conducted before and after an introductory workshop. The pre-evaluation aimed to catch the users's expected features and behaviours, while the main survey asked users about their concrete experiences with the NLS.

1.2.2.1 Pre-Evaluation

Responding to the first point (c.f. [#sec:expected-features]), the participants unanimously identified four of the six foundational language server capabilities that guided the implementation of the project (c.f. sec. **??**);

Type-information on hover was named almost uniformly. The participants showed a special interest in this feature describing specific behaviours. The desired information exposed by this feature are value types including applied contracts and documentation as well as function signatures.
When asked about the hover LSP method in particular, participants name additional function documentation, default values and the visualization of scopes as an additional features.

Diagnostics are widely understood as an important feature. Participant had very particular opinions about the behavior and detail of diagnostics including error message at the correct location in the code signaling syntax errors or possibly evaluation errors and contract breaches. In either case the diagnostic should be produced "On-the-fly" while typing or upon saving the document.
When asked about the diagnostics feature of language servers directly, the answers corroborated these initial opinions. In addition some participants named code linting, i.e. warnings about code style, unused variables, deprecated code and undocumented elements, as well as structural analysis hints as possible features. Structural analysis was imagined to go that far as being able to "suggest how to fix" mistakes in the code.

Code Completion was equally name in all but one response. It was described as a way to chose from possible completion candidates of options. The anwers included aspirational vague descriptions of such a feature including the a way to automatically prioritize specific items.

> Responding about the concrete LSP feature, participants listed variables, record fields, types, functions and function argument candiates as possible completion candidates. Moreover, some suggested the inclusion of the completion context to guide priorization as well as auto-generated contract and function skeletons.

Jump-to-Definition ~ was named significantly often. ~ The specific feature survey revealed the exoected behaviour in more detail; In general, the participants expect the feature to work with any kind of reference, i.e., variable usages, function calls, function arguments and type annotations. Record fields ale equally desired although the ability to define self referencing records was pointed out as a challenge. However, subjects expect statically defined nested fields to point to the correct respective definition.

1.2.3 Quantitative