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. **??** introduces methods employed for the evaluation. In particular, it details the survey (sec. **??**) which was conducted with the intent

to gain qualitative opinions by users, as well as the tracing mechanism (sec. **??**) for factual quantitative insights. Section **??** 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 answers 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 included in three fourth of responses. ~ 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.

The other two features Find-References and Workspace/Document Symbols on the contrary were sparingly commented. Participands noted that they did not use these capabilities. The features were however well understood, as shown by some responsess naming very particular distinctions of symbol types.

Beyond features that were explicitly targeted by this work, syntax highlighting and code formatting as well as error tolerance were named as further desireable features of a language server. Error tolerance was detailed as the capability of the language server to continue processing and delivering analysis of invalid sources restricting the computation to the correct parts of the program.

1.2.2.2 Experience Survey

1.2.2.2.1 Results The above figures show the turnout of three items from the survey for each of the relevant features. Neither of them shows clear trends with positive and negative reults distributed almost evenly between positive and negative sentiments.

The first graph (fig. **??**) represents the participants' general experience with the relevant features. It shows that each feature worked without issue in at least one instance. Yet, three features were reported to not work at all and no feature left the users unsurprised. Participant found the hover and diagnostic features to behave particularly unexpectedly.

For the second item of each feature, the survey asked the subjects to rate the quality of the language server based on their expectations. Figure **??** summarizes the results. Apart from the same three occasions in which a feature did not work for one participant, the majority of responses show that NLS met its user's expectations at least partially. The results are however highly polarized as the Jump-to-Definition and Hover features demonstrate; Each received equally many votes for being inapt and fully able to hold up to the participants expectations at the same time. Other features were left with with a nuniformly distributed assessment (e.g. Completion and Find-References). The clearest result was achieved by the Diagnostics feature, which received a slight but uncontended positive sentiment.

Asking about the general satisfaction with each feature, results in the same mixed answers. While a slight majority of responses falls into the upper half

of the possible spectrum, two features (of the three that have previously been reported without function) were given the lowest possible rating.

1.2.2.2.2 Comments

1.2.2.2.2.1 Hover As apparent in (fig. **??**), most participants experienced unexpected behavior by the LSP when using the hover functionality. In the comments, extraneous debug output and incorrect displaying of the output by the IDE are pointed out as concrete examples. However one answer suggests that the feature was working with "usually useful" output.

1.2.2.2.2.2 Diagnostics While the diagnostics shown by NLS appear to behave unexpectedly for some users in fig. **??**, all participants marked that those did not deter from keep using NLS for it as displayed in fig. **??**. In the comments some respondents praised the "quick" and "direct feedback" as well as the visual error markers pointing to the exact locations of possible issues while others mentioned "unclear messages." However, it was pointed out that it contracts were not checked by the Language Server. Moreover, a performance issue was brought up noting that in some situations NLS "queues a lot of work and does not respond."

1.2.2.2.2.3 Code Completion Comments about the Code Completion feature were unanimously critical. Some participants noted the little gained "value over the token based completion built into the editor" while others specifically pointed at "missing type information and docs." Additionally record field completion was found to be missing, yet highly valued.

1.2.2.2.3 Document Navigation Results and comments about the Go-To-Definition and Find-Referneces were polarized. On the one hand users reported no issues while others experienced unexpected behavior or were unabke to use the feature at all (cf. fig. **??**). Similarly, the comments on one hand suggest that "the feature works well and is quick" while on the other mention inconsistencies and unavailabilty. More practically, cross file navigation was named an important missing feature.

1.2.2.2.3.1 General Performance The responses to this item suggest that NLS' perfomance is largely dependent on its usage. On unmodified files queries were reported to evaluate "instantaniously." However modifying files caused that "modifications stack up" causing high CPU usage and generally "very slow" responses. Others pointed out that documentation was slow to resolve while the server itself was "generally fast."

1.2.3 Quantitative