

A study of which factors influence the choice for a static or a dynamic typing system

Carlos Souza, Eduardo Magno Figueiredo, Marco Tulio Oliveira Valente

¹Department of Computer Science, UFMG, Brazil

carlos.garcia@dcc.ufmg.br, figueiredo@dcc.ufmg.br, mtov@dcc.ufmg.br

Abstract. *Over the last years, dynamically typed languages such as Python, Ruby and JavaScript are becoming more popular in the software industry, previously dominated by statically typed languages. With more options, choosing the ideal language for each situation can become a complex task. In this paper we study which factors influence the choice of a programmer for a programming language when considering its typing system. We analyze the source code of more than one thousand projects written in Groovy, a language that allows programmers to choose between dynamic and static typing for each declaration. This analysis makes it possible to understand, for which types of declaration, programmers favor static or dynamic typing. Results show that Groovy programmers prefer static typing in most cases, specially when software maintenance is critical.*

1. Introdução

Linguagens de programação com tipagem dinâmica tem se tornado cada vez mais populares na indústria de software nos últimos anos. De acordo com o TIOBE Programming Community Index¹, um conhecido ranking que mede a popularidade de linguagens de programação, 27% das linguagens de programação adotadas na indústria possuem tipagem dinâmica. Em 2001, esse número era de apenas 17%. Entre as 10 linguagens no topo do ranking, 4 possuem sistemas de tipos dinâmicos: Ruby, Perl, Python e PHP. Em 1997, dentre essas linguagens, apenas Python e Perl apareciam no ranking, em 29º e 7º lugares respectivamente.

Linguagens com tipagem dinâmica possuem algumas vantagens sobre aquelas com tipagem estática. Por serem mais simples, os programadores conseguem executar suas tarefas de desenvolvimento mais rapidamente. Ainda, ao removerem o trabalho burocrático e repetitivo de declarar os tipos das variáveis, estas linguagens permitem que seus usuários foquem no problema a ser resolvido, ao invés de se preocuparem com as regras da linguagem [Chang et al. 2011, Tratt 2009].

Por outro lado, sistemas de tipos estáticos, também possuem suas vantagens. Estes conseguem prevenir erros de tipo em tempo de compilação. Pode-se argumentar que declarações de tipos aumentam a manutenibilidade de sistemas pois estas atuam como documentação, informando ao programador sobre a natureza de cada variável. Além disso, sistemas escritos a partir dessas linguagens tendem a ser mais eficientes, uma vez que não precisam realizar checagem de tipo durante a execução [Cardelli 1996, Pierce 2002, Bruce 2002].

¹www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Algorithm 1 Um exemplo de código escrito em Groovy. O método `agregateClasses` possui seu tipo de retorno, `ClassData`, declarado estaticamente. O parâmetro `classFilter` e a variável local `result` porém não possuem declaração de tipo. Ainda é possível observar o uso de um closure, outra característica dinâmica da linguagem, sendo passado como parâmetro para o método `findAll`.

```
ClassData agregateClasses(classFilter) {  
    def result = new ClassData()  
    classes.findAll(classFilter).each {  
        result += it  
    }  
    result  
}
```

Embora muito tenha se discutido a respeito das vantagens e desvantagens de sistemas de tipos estáticos ou dinâmicos, há poucas evidências a respeito da real influência destes sobre a evolução de sistemas de software. Alguns trabalhos [Hanenberg 2010, Prechelt and Tichy 1998, Daly et al. 2009] chegam a tentar analisar essa questão através de estudos controlados, mas esses não são capazes de reproduzir com precisão o contexto da indústria de software e portanto não são muito confiáveis.

Este artigo apresenta uma análise de como programadores usam sistemas de tipo em Groovy, uma linguagem com sistemas híbrido. Em Groovy, um programador pode escolher, para cada declaração, se utilizará tipagem dinâmica ou estática. Assim, espera-se que programadores escolham para cada situação o sistema de tipos mais adequado. Ao analisar como estes programadores usam sistemas de tipo, é possível entender quais fatores influenciam nessa decisão. Essa informação é capaz de ajudar desenvolvedores na escolha de linguagens de programação de acordo com o contexto em que se encontram. Projetistas de linguagens também podem se beneficiar desse conhecimento ao entenderem melhor como sistemas de tipos são usados.

1.1. Groovy

Groovy é uma linguagem dinâmica projetada para ser executada sobre a Java Virtual Machine. Sua sintaxe é parecida com a de Java, porém possui ela inclui funcionalidades dinâmicas, como metaprogramação e closures. É possível ainda escrever scripts em Groovy. Atualmente, Groovy ocupa a 50ª colocação no TIOBE Programming Community Index. Sua popularidade tem crescido bastante nos últimos anos, principalmente entre programadores Java que desejam incorporar algumas das facilidades de linguagens dinâmicas sem terem que aprender uma linguagem completamente nova ou mudar a plataforma de execução de seus sistemas.

Em Groovy, um programador pode escolher tipar suas declarações ou não. Tipagem estática e dinâmica podem ser combinadas no mesmo código livremente. É possível, por exemplo, definir o tipo de retorno de um método, mas manter os parâmetros deste método dinamicamente tipados. Graças à essa propriedade, é possível realizar uma análise sobre a preferência dos desenvolvedores com relação ao sistema de tipos para cada situação. O algoritmo 1 mostra um trecho de código escrito em Groovy.

2. Configuração do Estudo

O estudo apresentado neste trabalho consiste em analisar o uso de sistemas de tipos estático e dinâmico em um conjunto de projetos escritos utilizando a linguagem Groovy. A escolha por um destes sistemas é relacionada com fatores como o tamanho do projeto, tipo de declaração, visibilidade da declaração, entre outros. O objetivo é descobrir quais fatores os desenvolvedores consideram importantes na hora de escolher um sistema de tipos. Este estudo baseia-se na hipótese de que, dado que um programador é livre para escolher o sistema de tipos para cada declaração individualmente, ele irá sempre escolher o melhor sistema de tipos para cada situação encontrada. A seção 4 discute algumas ameaças à validade desta hipótese.

2.1. Dataset

Os projetos utilizados neste estudo foram obtidos do GitHub, um serviço de controle de versão baseado em Git bastante popular entre programadores Groovy. É possível obter do GitHub, de forma automatizada, o código de todas as versões dos sistemas sob estudo, assim como o histórico de commits e informações a respeito de cada desenvolvedor envolvido. Esta massa de dados é bastante significativa. São, ao todo, 1112 projetos abertos, totalizando 1676KLOC, considerando apenas a última versão de cada sistema. Estes sistemas foram desenvolvidos por 926 programadores e a idade de todos os projetos somada é de quase 300 anos. A figura 1 mostra a distribuição do tamanho dos projetos.

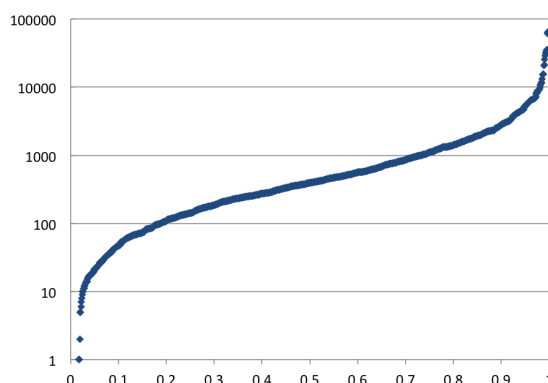


Figura 1. Distribuição dos 1112 projetos utilizados por tamanho do projeto em LOC.

2.2. Analisador Estático de Código

Para obter os sistemas de tipos utilizados em cada declaração foi implementado um analisador estático de código baseado na API de manipulação de ASTs² de Groovy. Esta API permite ao programador modificar em tempo de compilação o código gerado pelo programa. Em outras palavras, é possível programar o compilador. É possível ainda navegar pela AST gerada para cada uma das fases do compilador para um módulo Groovy. A fase escolhida para recolher informações sobre os sistemas de tipos utilizados foi a de Conversão. Nesta fase, já é possível acessar todas as declarações escritas pelo programador,

²<http://groovy.codehaus.org/Compile-time+Metaprogramming+-+AST+Transformations> (visitado em 10 de Dezembro de 2012)

Tabela 1. Declarações em Groovy

	Tipagem Estática	Tipagem Dinâmica
Geral	60%	40%

porem o compilador ainda não tentou resolver referências a outros tipos. Sendo assim, o analisador estático consegue analisar cada arquivo separadamente, sem precisar resolver nenhuma dependência externa.

Para cada classe ou script escrito em Groovy, são obtidas informações dos seguintes tipos de declarações

- Retorno de Método
- Parâmetro de Método
- Parâmetro de Construtor
- Campo
- Variável Local

Cada um dos tipos de declaração acima é agrupado por visibilidade, pública, privada ou protegida. É guardado ainda, para cada classe, se esta classe é um script ou a implementação de um teste automatizado.

3. Resultados

3.1. Resultado Geral

A tabela 1 mostra o valor relativo do uso dos sistemas de tipos considerando todas as declarações. Esta tabela mostra que, no geral, programadores preferem utilizar tipagem estática.

3.2. Resultados por Tipo de Declaração

A figura 2 mostra a quantidade relativa de declarações agrupadas por tipo de declaração. Variáveis locais utilizam tipagem dinâmica com mais frequência que outros tipos de declarações. Esse resultado já era esperado já que este tipo de variável possui menor escopo e ciclo de vida. Programadores não precisam se preocupar tanto em tipar essas variáveis já que elas não afetam outras partes do programa. Além disso, por serem criadas localmente, o tipo destas variáveis pode ser facilmente inferido por programadores que venham a trabalhar com essas variáveis.

Os tipos de declaração mais estaticamente tipados são os parâmetros de construtores. Pode se argumentar que há uma preocupação maior dos desenvolvedores em definir a interface dos métodos construtores uma vez que estes métodos definem a própria criação de um objeto e, caso não sejam invocados corretamente, podem comprometer o funcionamento desta instância. Outra possível explicação é que, como construtores tem o tipo de retorno tipado estaticamente por definição, programadores acabem tipando os parâmetros por coerência.

3.3. Resultados por Visibilidade

O gráfico da figura 3 mostra o uso de sistema de tipos por visibilidade de cada declaração. Repare que declarações públicas são estaticamente tipadas com mais frequência. Isto

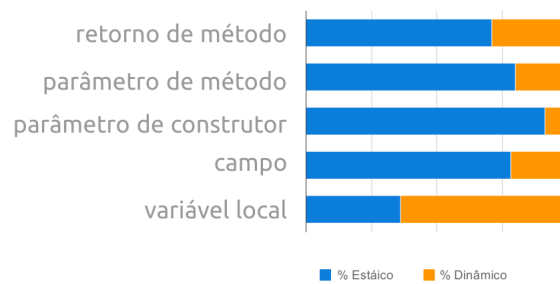


Figura 2. Sistemas de Tipo por Tipo de Declaração.

demonstra uma maior preocupação de desenvolvedores em tipar métodos e campos que compõem a interface de um módulo. Outro resultado interessante é que métodos protegidos são tipados com muita frequência. Em geral, métodos protegidos são utilizados como uma maneira de delegar parte da implementação necessária a uma super classe para a sua subclasse. Este tipo de interação possui acoplamento bastante alto. Assim, faz sentido para um programador tipar mais tais métodos.

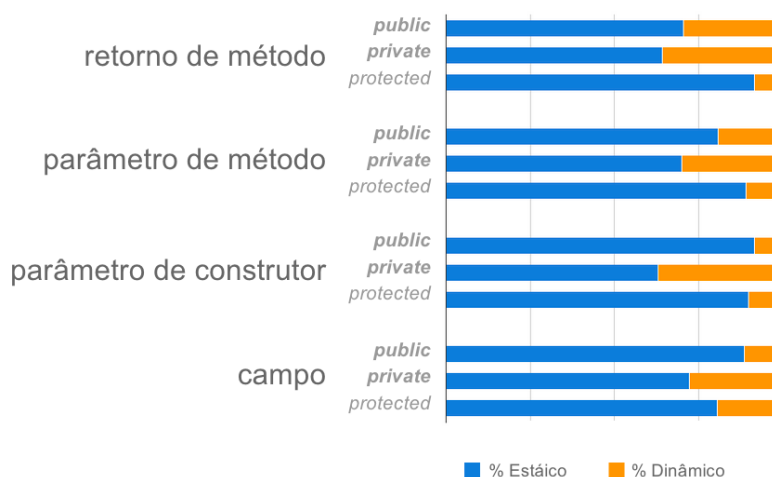


Figura 3. Sistemas de Tipo por Visibilidade da Declaração

3.4. Resultados por Tamanho de Projeto

O número de linhas de código é uma das métricas mais simples para definir a complexidade de um projeto. Foram correlacionadas as quantidades relativas de cada tipo de declaração com o tamanho de cada projeto. Assim, é possível entender se a complexidade de um projeto influencia no sistema de tipos escolhido pelo programador. A figura 4 mostra que existe uma correlação bastante significativa entre o uso de declarações tipadas em retornos e parâmetros de métodos públicos com o tamanho do projeto. Programadores tendem a se preocupar mais com a interface de seus módulos à medida que a complexidade do projeto cresce. Não foi possível observar nenhuma outra tendência significativa para outros tipos de declarações.

3.5. Scripts e Testes

As tabelas 2 e 3 mostram o uso de sistemas de tipos agrupados entre classes de teste/classes funcionais e classes/scripts. Aparentemente, classes de teste e scripts não influenciam

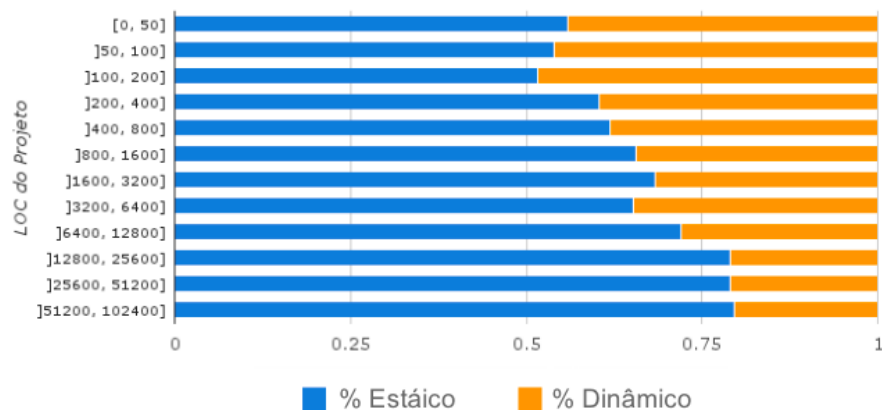


Figura 4. Sistemas de tipo de retornos e parâmetros de métodos públicos agrupados por tamanho de projeto. O ranking de Spearman para a relação entre tamanho de projeto e uso de declarações com tipo para este tipo de declaração é de 0.95.

Tabela 2. Declarações em Groovy agrupadas por classes de teste e classes funcionais

	Tipagem Estática	Tipagem Dinâmica
Classes de Teste	57%	43%
Classes Funcionais	62%	38%

na escolha pelo sistema de tipos.

4. Ameaças à Validade

Este trabalho se baseia na hipótese de que programadores irão escolher, para cada situação, o melhor sistema de tipos. Essa hipótese porém não é muito robusta. Em primeiro lugar, não existe um conceito único do que é melhor para cada situação. Por exemplo, um programador pode entender que declarações em classes de teste precisam ser tipadas uma vez que teste automatizado atua também como documentação. Outro programador pode preferir declarar variáveis sem tipo em classes de teste pois acredita que código de teste, por não agregar valor funcional, deve ser simples e rápido de se escrever.

Há diversos fatores difíceis de se medir que podem influenciar a escolha de tipos. Alguns frameworks exigem que o programador use um determinado sistema de tipos. Uma das maneiras de se criar mocks usando o framework de testes Spock³, por exemplo, exige que o programador declare seu mock usando tipos estáticos. Programadores podem ter maior experiência em linguagens com um determinado sistema de tipos e continuar a

³spockframework.org

Tabela 3. Declarações em Groovy agrupadas em classes e scripts

	Tipagem Estática	Tipagem Dinâmica
Classes	61%	39%
Scripts	54%	46%

usando esse mesmo sistema de tipos em Groovy. Em particular, programadores Java que começam a usar Groovy pela semelhança entre as linguagens continuam usando tipos em suas declarações com bastante frequência.

Dadas as ameaças listadas acima, este trabalho evita tecer quaisquer tipos de conclusões definitivas a respeito de qual sistema de tipos é melhor para cada situação. Ao invés disso, são discutidas possíveis causas para cada resultado observado

5. Related Work

To the best of our knowledge, there are no studies on the literature that try to answer the question of which are the factors that drive the choice of a programmer for a static or a dynamic typing system. Some studies, however, present a comparison between these two.

In [Hananberg 2010], the author compares the performance of two groups of students when asked to develop two small software systems. Both groups used a language designed by the authors, Purity, the only difference being that one group used a statically typed version and the other a dynamically typed one. The results show that the group using the dynamically typed version of Purity was significantly more productive. As in our study, the authors were able to compare the two typing systems directly, in this case by designing their own language, thus eliminating any external factors in the analysis. However, it can be argued that these results might not represent the real life in the industry precisely since this was a small duration study, considering programmers as examples of real life programmers, and with no interaction with other programmers. We try to provide more reliable results by analysing actual source code developed by programmers in their normal activities.

In a continuation of the previous study [Kleinschmager et al. 2012], the authors came to opposite conclusions. They compared the performance of two groups in maintenance tasks, one using Java, a statically typed language, and the other, Groovy, enforcing the use of dynamically typed declarations only. In this case, the group using Java, the statically typed language, was much more productive. These contradictory results reinforce our argument that studies based on source code developed at the industry might yield more reliable results than results obtained in controlled studies.

Results similar to the previous study were obtained in [Prechelt and Tichy 1998, Gannon 1977]. These studies however are very old and do not consider newer Software Engineering tools and practices, which can help programmers to avoid some of the common errors in dynamically typed software. In our study, the oldest source code dates from 5 years ago, which ensures that we don't share the same problem.

In the experiments performed in [Daly et al. 2009], the authors study two groups of programmers on small development tasks. One group used Ruby, while the other used DRuby, a statically typed version of Ruby. Results showed that the DRuby compiler rarely catches mistakes that were not already evident to programmers. Most of the programmers however had previous experience with Ruby and were probably used to the lack of typing in their declarations, influencing the outcome of this study. In our study a similar question can be raised. Since most of Groovy programmers have some previous experience with Java, they might tend to keep using types in their declarations.

6. Conclusão e Trabalhos Futuros

Este trabalho caracteriza como sistemas de tipos são utilizados em Groovy. Resultados mostraram que declarações com tipo são, em geral, mais frequentes na maioria dos casos. O único tipo de declaração onde o uso de sistema de tipos dinâmico é mais comum são as declarações de variáveis locais, onde o escopo limitado e a proximidade da criação de uma variável fazem o uso de declarações tipadas muitas vezes desnecessário. Não há diferença significativa no uso do sistema de tipos entre classes e scripts ou entre classes funcionais e classes de teste.

De forma geral, pode se observar que o uso do sistema de tipos estático é mais frequente em declarações mais importantes para o correto funcionamento do sistema, tais como métodos e campos públicos, que definem a interface de um módulo, ou métodos protegidos, que definem um contrato com alto acoplamento entre uma classe e suas subclasses. Foi possível ainda visualizar um crescimento no uso de declarações tipadas em métodos públicos à medida que o tamanho dos projetos cresce. Tal observação é um indício de que programadores consideram declarações tipadas mais robustas.

Este trabalho é apenas a primeira parte de um trabalho mais completo, cujo objetivo é analisar como sistemas de tipos influenciam o desenvolvimento de software. Em particular, deseja-se responder as seguintes questões:

- Variáveis declaradas sem tipo possuem maior correlação com o aparecimento de defeitos no sistema?
- Variáveis declaradas sem informação de tipo prejudicam a manutenibilidade?
- Quais fatores influenciam na decisão pelo sistema de tipos?
- É possível construir grandes sistemas a partir de linguagens com sistemas de tipo dinâmicos?

O dataset e a metodologia utilizados neste trabalho serão utilizados em trabalhos futuros para responder as questões acima. Há ainda a necessidade de desenvolver novos tipos de análises, como o estudo do histórico do sistema e o relacionamento entre o uso de sistemas de tipos e métricas de software.

Referências

- Bruce, K. (2002). *Foundations of object-oriented languages: types and semantics*. MIT press.
- Cardelli, L. (1996). Type systems. *ACM Comput. Surv.*, 28(1):263–264.
- Chang, M., Mathiske, B., Smith, E., Chaudhuri, A., Gal, A., Bebenita, M., Wimmer, C., and Franz, M. (2011). The impact of optional type information on jit compilation of dynamically typed languages. *SIGPLAN Not.*, 47(2):13–24.
- Daly, M. T., Sazawal, V., and Foster, J. S. (2009). Work In Progress: an Empirical Study of Static Typing in Ruby. In *Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU)*, Orlando, Florida.
- Gannon, J. D. (1977). An experimental evaluation of data type conventions. *Commun. ACM*, 20(8):584–595.
- Hanenberg, S. (2010). An experiment about static and dynamic type systems: doubts about the positive impact of static type systems on development time. *SIGPLAN Not.*, 45(10):22–35.
- Kleinschmager, S., Hanenberg, S., Robbes, R., and Stefik, A. (2012). Do static type systems improve the maintainability of software systems? An empirical study. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 153–162.
- Pierce, B. (2002). *Types and programming languages*. MIT press.
- Prechelt, L. and Tichy, W. F. (1998). A controlled experiment to assess the benefits of procedure argument type checking. *IEEE Transactions on Software Engineering*, 24:302–312.
- Tratt, L. (2009). Chapter 5 dynamically typed languages. volume 77 of *Advances in Computers*, pages 149 – 184. Elsevier.