Mayer *et al. Journal of Software Engineering Research and Development* (2017) 5:1
DOI 10.1186/s40411-017-0035-z

Journal of Software Engineering Research and Development

RESEARCH Open Access

On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers

Philip Mayer[1*], Michael Kirsch[1] and Minh Anh Le[2]

*Correspondence:
mayer@pst.ifi.lmu.de
[1]Programming & Software Engineering Group, Ludwig-Maximilians-Universität München, München, Germany
Full list of author information is available at the end of the article

Abstract

Context: Non-trivial software systems are written using multiple (programming) languages, which are connected by cross-language links. The existence of such links may lead to various problems during software development. There is little empirical evidence on the incidence of these problems and the experiences of professional developers in this field.

Aim: We want to provide empirical evidence on multi-language software development, cross-language linking, and tool support in industry, including the views of professional developers on benefits and problems in these areas.

Methods: We conducted a survey study to gather responses from 139 professional software developers.

Results: Respondents reported an average of 7 languages and 3 linked language pairs per project. Respondents saw benefits of multi-language development for the motivation of developers and the translation of requirements, but problems in understandability and changeability. Over 90% of respondents reported problems related to cross-language linking. Developers universally agree on the usefulness of tool support.

Conclusions: Multi-language programming and cross-language linking seem common but lead to several problems. We suggest that future practical as well as research efforts focus on these issues by creating appropriate tool support and by developing better techniques for cross-language linking for improved changeability and understandability.

Keywords: Multi-language software development, Cross-language links, Tool support, Survey, Professional developers

1 Introduction

Non-trivial software systems are not written in just one programming language. Instead, multiple languages are used; among these are the usual general-purpose programming languages (GPLs) like Java, C#, or Ruby, but also domain-specific languages (DSLs) such as SQL, HTML, or configurable languages such as XML. A recent survey of open source projects has shown that the use of multiple languages is rather universal, with

a mean number of 5 languages used per project (Mayer and Bauer 2015). Thus, *multi-language software development* (MLSD) seems to be common, at least in the open source world.

When using multiple languages for the development of one project, these languages usually encode different *aspects* of a system, that is, the individual code artifacts do not stand alone but are in fact connected by some means. In many cases, common identifiers (names) are used for creating *links* between the individual code artifacts and thus between the languages used. An example of a typical link is shown in Fig. 1, where the identifier `textfield` is shared between Java and HTML and used to retrieve a user input value in a web application.

The use of such *cross-language links* (abbreviated here to XLL) — that is, points in the system where code in two languages is connected — may be part of an explicit interface specification such as when using JNI between Java and C. In many cases, however, such links exist between GPLs and DSLs and, as in the example, are more implicit, being spread through the code as required by a certain framework or library.

Such links may present problems to developers. First, they are usually not part of any of the languages themselves but rather stand outside, which means that they are not checked at design time for correctness, but will fail at runtime if specified incorrectly if not for additional tool support. Second, identifiers used for cross-language purposes are not always dedicated only to this purpose, but fulfill other roles; for example, consider a Java class name referenced from XML. Later changes to the system might include renaming the class, which in turn would require changing links to this identifier as well, which may or may not happen. Third, knowing that there may be links in the system — but not exactly where — might even lead developers to refrain from changing identifiers for fear of breaking the system due to unknown side effects of changes. This leads to a degradation of software quality and thus to maintenance problems (technical debt).

Several researchers have shown cross-language links to be a factor in real-life software applications; see for example (Mayer and Schroeder 2014; Pfeiffer R and Wasowski 2012b; Favre et al. 2012) with proposals for aiding developers when dealing with such links, mostly by tool support, but also by pro-active modeling efforts.

Software development in general is supported by a myriad of *tools*, ranging from command-line compilers to fully integrated development environments. Tools exist that specifically support different functions for dealing with cross-language links,

Listing 1: HTML Code

```
1 <form action="sayHelloServlet">
2 Your name: <input type="text" name="textfield" />
3 <input type="submit" value="sayHello"/>
4 </form>
```

Listing 2: Java Code

```
1 String name= request.getParameter("textfield");
2 PrintWriter out= response.getWriter();
3 out.println("Hello " + name);
```

Fig. 1 Cross-Language Linking (XLL) example

such as highlighting, error marking, or even automated rename refactoring. Dedicated tools for cross-language linking become important since the usual infrastructure of compilers and editors is mostly focused on supporting individual languages. Often, there is support for *using* multiple languages within a development environment — for example through plugins — but support for interaction points (cross-language links) between the languages is another issue. Existing work on the existence and usefulness of cross-language tool support to developers is available (e.g. Pfeiffer and Wasowski (2015); Pfeiffer and Wasowski (2012a)).

In this study, we will be concerned with all of the three topics introduced above — multi-language software development in general, cross-language linking, and tool support. While there is existing work in each of the three areas, there are — to the best of our knowledge — no survey studies on these topics (industrial or otherwise) with one exception (a survey of language developers which touches the topic of XLL Pfeiffer and Wasowski (2015)). It is our aim to remedy this situation and to provide an industrial perspective based on the experience of professional software developers.

We believe that this perspective provides important insights which can serve as enabling data for both practical development and future research of methods, techniques, and tools for software development with multiple languages and cross-language links. In particular, besides numerical data such as how many and which languages were used, how many and which languages were linked, and which tool support functionality was available, we also ask developers about their views on benefits and problems: Do they feel multi-language software development in general is beneficial or detrimental to their work? Did problems occur in cross-language linking and if so, which, when, and how frequent? What was done about them? Do they feel tool support is beneficial for their work, and if so, which functions are most needed?

We gather data for this study by means of a survey; more precisely, an (online) questionnaire. As a guide for the design of the questionnaire itself and for the evaluation, we have posed six research questions in three categories which mirror the three areas discussed above. Within each category, the first question is about "hard" data such as numbers and names of languages, links and tools, while the second is about developer opinions.

On multi-language programming

- RQ1: How prevalent is multi-language programming and which languages are used? How many languages did developers work with?
- RQ2: What are the benefits and problems developers encountered in the use of multiple languages? Do developers feel that multi-language programming has/will increase or decrease over time?

On cross-language linking

- RQ3: In how many and which combinations of languages did developers encounter cross-language links?
- RQ4: Did problems with cross-language linking occur? If so, which, when, how frequent, and what was done to alleviate these problems?

On tool support

- RQ5: Was tool support available for dealing with cross-language identifiers? If so, which functions were available?
- RQ6: Is tool support considered important, and if so, which functions for handling cross-language identifiers are most important?

The result and discussion sections of this paper will be structured along the lines of these three categories and research questions. Additionally, we investigated the metadata, that is, attributes such as experience of the respondents of the survey, including project data (such as team size and duration). This data is cross-cutting across all of the research questions above and serves to illuminate our sample.

We outline the methods used in this study — namely, questionnaire design, participant selection, and the methods used for evaluating the results — in the next section. Sections 3 and 4 contain the results from our analysis and a discussion of the implications and conjectures we can draw from the data, respectively. Related work is discussed in Section 5, and we conclude in Section 6.

2 Methods

This study uses the survey research method. A survey may serve different goals and is usually categorized as descriptive, explanatory, or explorative (Wohlin et al. 2012). This survey is mostly descriptive and in some parts explorative — that is, we do not try to verify previous hypotheses but attempt to describe the situation as it presents itself to our respondents. Some parts can be seen as explorative since little empirical research has been done on this topic before and there are indeed several areas where we did not know what to expect, with the results thus forming a baseline for future research. As the method for data collection, we use an online questionnaire. We describe this questionnaire in Section 2.1.

The questionnaire was sent out to participants from industry which had to be selected beforehand and invited via e-mail along with a reminder and a follow-up on participation. We describe this process in Section 2.2.

The answers collected from the participants include quantitative and qualitative results. The former were analyzed using (descriptive) statistical methods; the latter by interpretation, which we discuss in Section 2.3.

2.1 Questionnaire design

The structure of the research questions shown in the previous section is reflected in the questionnaire which contains the same three sections. An additional section was used to gather metadata. We followed the guidelines from (Kitchenham and Pfleeger 2002b) in the design of the questionnaire.

When looking at the research questions it is clear that many questions can only be answered sensibly with regard to a concrete software system or *project*, while others are more general and the developers should be allowed to draw on their entire *professional experience*. For example, the languages or cross-language links used should be answered for a concrete project, while the opinions on the general benefits or problems of multi-language software development should be independent from a concrete project. Developers were asked to think of their *last completed project* in the beginning

of the questionnaire, and answer all project-related questions with regard to this particular project.

The answering scales for each question were designed in accordance to best practices from the social sciences Franzen (2014). We used odd counts of answers for scales and between 3 and 9 options. All answering options carried explicit names, and were designed to be disjunct and complete. In most questions we added a text field to allow participants to add their own free text instead of selecting an existing option.

The questions asked are provided, in an abbreviated form[1], in Section 2.1.1 (Table 1). This table is split into four sections. The first three correspond to the three categories of our research questions; the fourth contains questions about demography and attributes of the last completed project. In the online questionnaire, each section is prefixed with an explanatory text which clarifies what the section is about, explains the terms used, and gives examples.

2.1.1 Section 1: multi-language software development

In the first section, we asked questions on the general topic of multi-language software development. An introductory text was shown to participants in the beginning which defines the terms "general-purpose" and "domain-specific", including examples. All but the last two questions are directly related to the last finished project of participants. As can be seen from the table, the first two questions relate to which general-purpose languages were used and which DSL types were present. These questions serve, in a double function, as a "gentle introduction" into the topic by providing multiple-choice answers with explanations of the terms GPL, DSL, and DSL type in the context of this study.

Question **1** asks about GPLs. These languages are typically well known at least by name, and their number is rather small; we suggested 11 languages which are the most-used languages from (Mayer and Bauer 2015).

For DSLs, the situation is different, as there are a multitude of options to be considered. In order not to overwhelm participants, we thus opted to ask for the use of any language of a *DSL type* in question **2**. We suggested seven types which were identified in (Mayer and Bauer 2015) – namely UI, Shell, Build, Configuration, Querying, Rule Specification, and Parsing/Lexing.

The third question (**3**) asks whether a custom language was created specifically for the project, and if so, what its purpose was (as free text).

Question **4** is the last related to concrete languages. Here, we asked how many of the project's languages were used (as in *changing code*) by individual developers. Respondents were asked to move a slider in 9 steps. On the left side, the selection was "one language"; on the right side, it was "all languages". The middle corresponded to "about half of the languages".

The last two questions do not relate to the respondents' last projects but to their general opinion on multi-language software development. Question **5** very generically asks whether respondents feel that the use of multiple languages in a software project is beneficial or detrimental to various tasks, such as translating requirements to code, system understandability, or developer motivation. Respondents could select five options for each aspect, which were "very beneficial", "rather beneficial", "neutral", "rather

Table 1 Abbreviated survey questions and possible answers

No.	Question	Answer format	Options/suggestions
Section 1: Multi-Language Software Development (MLSD)			
1	Which GPLs were used in your last project?	Multiple choice	C; C++; C#; Java; ...; other (free text)
2	Which DSL types were used in your last project?	Multiple choice	UI languages, shell languages; build languages, ..., other (free text)
3	Was a custom language used?	Yes+free text / no	
4	How many of the project's languages did developers write code in?	Scale 1-9	Slider with 9 options from "one language" to "all languages"
5	Is MLSD beneficial or detrimental for these aspects of software development?	Scale 1-5	Requirement translation, architectural design, first implementation, changeability, understandability, build management, CPU performance, memory requirements, developer effort, developer motivation (Scale: 5 options each from "very detrimental" to "very beneficial")
6	Were there/will there be more/less languages in the past/future?	Scale 1-3	Past / future with three options each: fewer, same, or more languages
Section 2: Cross-Language Links (XLL)			
7	Which links between GPLs and DSLs were used in your project?	Yes+free text / no	Free text
8	Which links between GPLs and GPLs were used in your project?	Yes+free text / no	Free text
9	Which links between DSLs and DSLs were used in your project?	Yes+free text / no	Free text
10	Which problems with cross-language links were encountered?	Multiple choice	Bugs due to changes; configuration issues; change avoidance; build complexity; understandability/communication issues; harder to write unit tests; other (free text)
11	*When* did problems occur, and how frequently?	Scale 1-5	First implementation; changes due to new requirements; changes due to refactorings; unit testing; test phases (by dedicated personnel); user testing; after release (Scale: 5 options each from "did not occur" via "from time to time" to "all the time")
12	Which measures were taken to prevent cross-language linking problems?	Multiple choice	Avoided MLSD; avoided XLL; avoided changes in XLL; special care when changing XLL; dedicated tools, dedicated tests; other (free text)

Table 1 Abbreviated survey questions and possible answers *(Continued)*

Section 3: Tool Support			
13	Which cross-language tool functions were available?	Multiple choice	Highlighting; error marking; navigation; refactoring; other (free text)
14	How important are these tool support functions?	Scale 1-5	Highlighting; error marking; navigation; refactoring; other (free text) (Scale: 5 options from "very unimportant" to "very important")
Section 4: Metadata and Demographic Questions			
15	How many years have you worked in professional software development?	Free text	
16	Which responsibilities did you have in your last project?	Single choice	Developer, tester, operations, ..., other (free text)
17	How many software developers worked in your last project?	Free text	
18	How long was the development phase of your last project? (in months)	Free text	
19	Which software category does your last project fit?	Multiple choice	Client/Server, web applications, server only, ..., other (free text)
20	Do you have any additional comments?	Free text	

detrimental", and "very detrimental". It was also possible to add free text, although without the option to vote.

Question **6** asks for a subjective assessment of whether there were more or less languages per project in the past, and whether participants think that there will be less or more in the future. Possible answers for each were "fewer", "same", and "more".

2.1.2 Section 2: cross-language links

Section two is the most complex section of the questionnaire. At its start, an explanation of cross-language links was shown to participants (including Fig. 1 as an example of a GPL/DSL link). We also gave examples for GPL/GPL links (the Java JNI functionality) and DSL/DSL links (the very common use of CSS class names in HTML). This introduction served to prepare respondents for the following questions.

The first three questions (**7**, **8** and **9**) then ask whether there were links between languages in the participant's last project — individually for GPL/DSL-links, GPL/GPL-links, and DSL/DSL-links. For each, participants were asked to provide their language pairs as free text. While this type of answer is more laborious for participants, it offers the ability to participants to easily add additional languages or types. The free text answers provided here had to be manually coded during data analysis.

The last three questions of section two ask about problems that occurred when using cross-language links, again in the last project. These questions were only asked if participants indicated that they in fact used cross-language links, i.e. they answered yes in one of the first three questions. Otherwise, the questionnaire directly proceeded to the third section.

Question **10** is about the types of problems that occurred when using cross-language links; examples are problems with framework configurations, build errors, problems creating tests, or errors when renaming identifiers. Question **11** asks *when* these problems occurred (for example, during first implementation, during refactoring, during testing). Respondents were asked to select the frequency of occurrence on a scale between "never" and "all the time" for each of the options.

The last question (**12**) was about possible measures that were taken to prevent problems with cross-language links, such as using dedicated tools or specialized tests, or even attempting not to use multiple languages or cross-language links in general.

2.1.3 Section 3: tool support

The third section is about tool support for cross-language linking. Again, respondents were shown an introductory text which explains tool support for cross-language linking including the four functions we used as suggestions in the answers below.

There are just two questions in this section. The first (**13**) is, as usual, related to the last project and asks whether the tools used by participants offered one of the four typical functions (highlighting, error marking, navigation, and refactoring) that are commonly mentioned in the literature; additionally, a free text field was provided. Respondents were instructed to select a function even if it was only present for some of the languages used. It was also possible to state that no functions were available at all.

The last question (**14**) is again independent from the last project. In this question, we asked whether tool support in general and for each of the four functions mentioned above individually is seen as beneficial or detrimental. In each case, respondents

were asked to select on a scale from "very unimportant" via "neutral" to "very important".

2.1.4 Section 4: demographic and other questions

In the final section of the questionnaire we gathered some general information about the participant and his/her last project to be able to paint a picture of the sample of participants we have taken.

The first question (**15**) is about the years of professional experience of the participant. The next questions are all related to the last project of the participant. We asked about the responsibilities of participants (question **16**), about the team size and the duration in months of the last project (questions **17** and **18**), and finally about the type of the system developed (**19**). Here, we provided six suggestions with the usual classification (Web, Client/Server, Server Only, Client Only, Mobile, and Embedded).

The final question **20** allowed respondents to add their own free comments to the overall topics in the questionnaire.

2.1.5 Summary

In order to keep the dropout rate low, the questionnaire was designed to be answerable in 15 mins. In total, there was a maximum of 20 questions; as indicated above, some might be skipped due to the participant's answers, shortening the time to completion.

Since most of our participants (see next section) are native German speakers, the questionnaire was created in German and later translated to English. Every participant could freely choose to complete the questionnaire in either language.

2.2 Participant selection and study execution

A central goal of this study was to survey *professional developers*, that is, people who earn their living in industry developing software for clients or their own company. Unfortunately, compared to students or even open source developers, this created the problem of acquiring participants in the first place, let alone creating a random selection of a well-defined population. It was also unclear whether sending e-mails to non-consenting recipients would be acceptable under German privacy and unsolicited e-mail laws.

Due to these issues, we used an opt-in approach to study participation, that is, participants were asked beforehand by personal contact whether they would like to participate in the study. Besides the prevention of legal issues, this also ensured that all participants were indeed industrial developers. Thus, participants were basically recruited by the study authors and colleagues: A total of 12 people including the first two authors managed to invite 194 developers who explicitly agreed to taking part, all of which were actively working in the software engineering field in industry at the time. Nearly all work in Germany, and most work in the Munich area. There were only a few exceptions, who took part from Sweden and the United States.

As will be seen in the next section, the 139 participants who eventually responded work in over 20 companies, have between 1 and 36 years of experience, and reported project sizes from between 1 to 170 participants and 1 to 60 months duration. We thus believe the results from this study to be fairly meaningful. Nevertheless, the selection process as outlined above is a form of snowballing (Kitchenham and Pfleeger 2002a). We thus

refrain from using inferential statistics and restrict ourselves to describing the sample we have taken.

The study was executed by using the web application *SoSciSurvey*[2] which is dedicated to the execution of such questionnaires (originally for the social sciences). SoSciSurvey allows study execution in a fully anonymous fashion while still being able to track participants by generating tokens for each participant and automatically sending these out by e-mail. The mapping from e-mail address to token is handled internally and never exposed to the conducting researcher.

Using tokens ensured that no participant took part twice and prevented answers by non-qualified participants (i.e., non-developers) while keeping anonymity. It also allows us to identify who had responded to the survey (without seeing their answers), which has the benefit of being able to send reminder emails and report, in a limited fashion based on e-mail domains, which companies took part in the survey.

A time frame of two weeks in July 2015 was set for participants to answer the survey. Two emails were sent to participants: One in the beginning, and one after one week to those who had not yet answered the questionnaire. Recipients had the option to provide their e-mail address at the end of the questionnaire for receiving the study results. The e-mail address was stored separately from the answers.

After the end of the survey period, the complete data was downloaded from SoSciSurvey and analyzed offline.

2.3 Coding and statistical methods

Where the questions in the questionnaire do not have a free text answer, we use either metric scales (such as for the years of experience), ordinal scales (such as a value between "very detrimental" and "very beneficial"), nominal scales (such as types of systems) or binary scales (yes and no). For answering our research questions, we can use the standard methods of descriptive statistics, that is median, the quantiles, and the accompanying visualization methods, in particular (stacked) barplots. This allows us to summarize the data from our sample to better understand it. All analyses have been performed using R (Core Team 2015). The code is available online[3].

Questions seven to nine required answers as free text, which had to manually coded (i.e. converted into appropriate language pairs) before they could be analyzed quantitatively. Participants provided answers such as "HTML to CSS" or "Java+JS", which we manually transformed into a consistent, countable format ("HTML$\longleftrightarrow$CSS", "Java$\longleftrightarrow$JavaScript", etc). These pairs were then counted automatically using a simple Java program (which is available on our web page as well).

In several other questions we allowed free text answers *in addition to* suggested multiple choice answers. These were not formally encoded; instead, where appropriate, we have included these results *qualitatively* in Section 3 when discussing the quantitative results of the corresponding questions. The comments in question 20 were added to the questions they referred to in this fashion as well.

3 Results

This section presents the results from our survey. Section 3.1 lists the results from the participant selection process and the answers to the questionnaire section four (metadata

and demography). We then discuss the three main sections of the questionnaire in turn in Sections 3.2, 3.3 and 3.4.

3.1 Profile of the participants

As indicated in the previous section, we had a total of 194 people who explicitly opted in to take part in the study. All of these received an e-mail with a link to the questionnaire. In total, 159 participants opened up the survey web site; 150 answered at least one question, and 139 participants completed the survey (i.e. answered all questions); thus, the return rate is 72%.

When looking at the e-mail address domains of respondents, we find that 88 respondents came from 23 identifiable companies. The remaining 51 respondents had e-mail adresses with private domains or were using e-mail service providers (such as `gmail.com`).

We now turn to the actual answers respondents have given in section four of the questionnaire. We begin with the industrial experience of respondents. Figure 2 shows the distribution of the years of experience of developers.

The median number of years is 8, with 50% of respondents having between 5 and 15 years of experience. One person had less than six months, one person 36 years of experience (note that we asked for the *year in which participants started working*; combined with the survey being performed in July, 0 years thus actually means up to 6 months; 1 year means from 6 months to 18 months, and so on).

The other questions in this section relate to the last finished project of a participant. The first relates to the responsibilities the respondent performed as part of his or her job on the project. Figure 3 shows the result, in which respondents were allowed to select all responsibilities that applied. 88% of respondents directly worked as programmers on the code, with about 75% being involved in design and 54% in testing. About 6% filled in their own responsibilities, which were related to requirements (6 responses) and product management (2 responses).

In the next two questions we investigated the size of the project by asking for the number of team members involved in development and the number of months of the development phase.

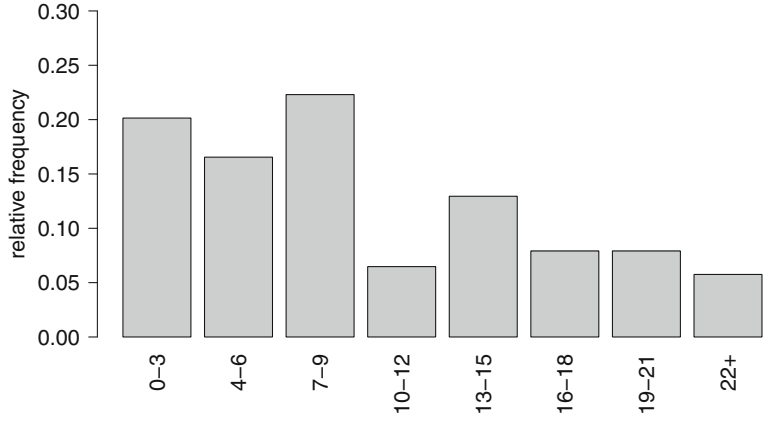

Fig. 2 Professional experience of the respondents in years (0: up to 6 months) (categorized)

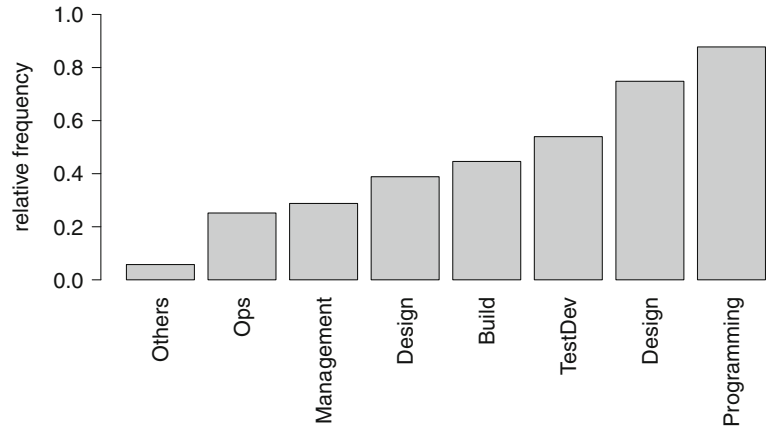

Fig. 3 Job responsibilities of participants (multiple choice)

The team size of projects ranged from 1 to 170; however, the median is 7 and 50% of projects had between 4 and 18.5 developers. If we take the inner 80% (10 and 90% quantiles) we get a range of 3 to 60 developers.

The number of months ranged from 1 to 120, with a median of 12 and 50% between 6.5 and 24 months, that is half a year to two years with a median of one year. If we take the inner 80% we get a range of 3 to 40 months.

The final question in section four is about the type of software system that was developed. This was a single-choice answer. As shown in Fig. 4, about 36% of respondents indicated a web application, with client/server following close behind (30%). These systems thus make up 66% of responses. Server only follows behind with 16%. The other four categories were selected by less than 10% of respondents each.

3.2 Section 1: multi-language software development

In this section, the first three questions relate to the languages used in the last finished project of respondents. Based on the answers, we can both identify the individual languages (or language types) used as well as the number of languages selected.

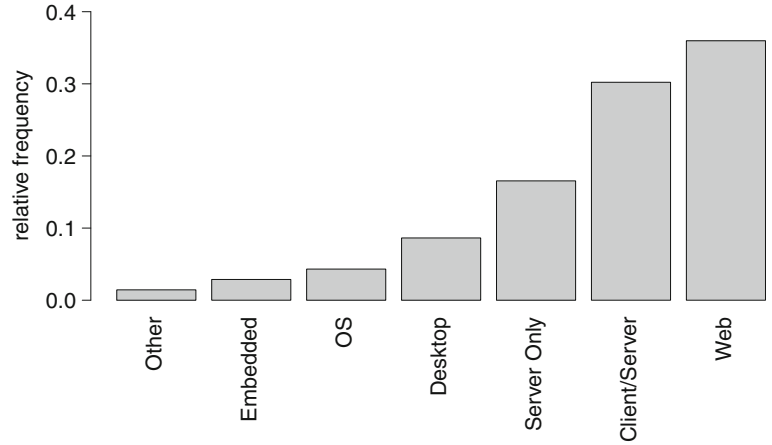

Fig. 4 System type of the last project (single choice)

The first question is about the general-purpose languages used. In addition to our suggestions, respondents could add their own by using a free text field. Three additional languages were mentioned three or more times each (VisualBasic, TypeScript, and Groovy) and thus were coded as variables as well.

The results are shown in Fig. 5. Only 5 languages were selected by more than 10% of respondents. Java is the dominant language in the sample with 76%, followed by JavaScript with 56%. The three others are Python (17%), C# (12%), and Ruby (10%). The full data set is available on our web site[4].

The median of GPLs used is 2, with 50% of projects using between 2 and 3 GPLs. Most projects (inner 80%) use between 1 and 4 languages.

The second question asks for the use of languages from a set of *DSL types*. The answers to this question (Fig. 6) show a major difference between the last five choices and the first three. Each of the last five was selected by over 75% of respondents; thus, most projects used at least one language from each of these areas. By comparison, the other three (rule languages, parsing/lexing, and others) were selected by under 10% of recipients. Besides the seven suggestions, recipients could also add their own type; among those mentioned were code generation languages (like Velocity) and transformation languages (like XSLT).

Thirdly, we asked whether a custom language was developed in the project as well. A total of 12 respondents (9%) reported such a custom language. The categories of custom languages were very diverse; we see mentionings of languages for the user interface, state management, configuration, querying, and code generation.

We have now seen three questions on the use of programming languages. In order to understand how many languages were used in a project, we can add these language counts together. However, as we asked for DSL *types* instead of DSLs themselves, this will be a lower bound on the number of languages used. The result shows a median of 7 languages per project, with 50% of projects having between 6 and 8 languages each. Most projects (inner 80%) have between 4 and 9 languages.

The last question relating to language use was about the number of languages an individual developer was working on in the project. The result (Fig. 7) shows that there is no clear answer to this; in fact all possible options (from “one language” to “all languages”) were chosen by between 6% and 15% of respondents.

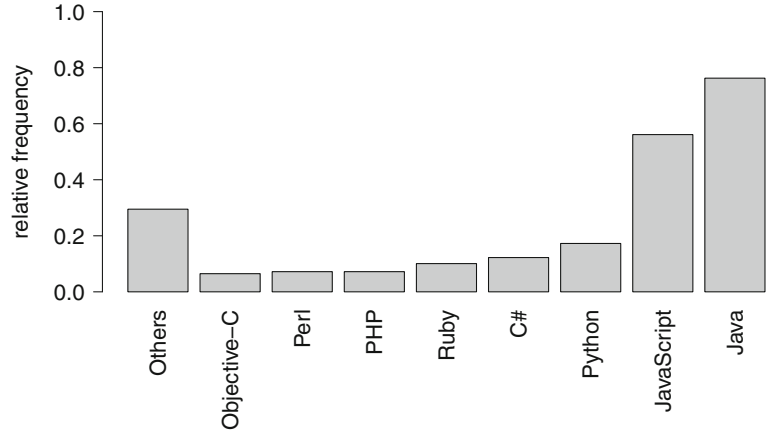

Fig. 5 General-Purpose Languages (GPLs) used in the projects of the respondents

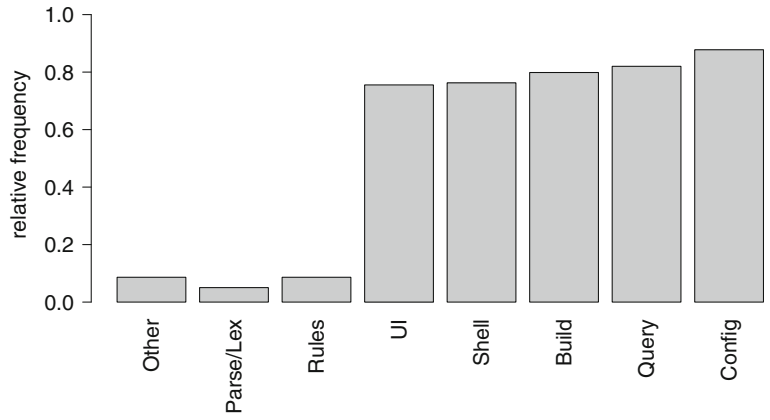

Fig. 6 Domain-Specific Language (DSL) types used in the projects of respondents

Questions five and six of section one are not related to a respondent's last project, but to their general experience and view on the use of multiple programming languages. In question five, we asked whether developers view the use of multiple languages *in general* as beneficial or detrimental for 10 suggested aspects of software development (plus free text). The results are shown in Fig. 8. The figure shows 10 different aspects on the y axis. The x axis contains the relative frequency; from black on the left ("very detrimental") to white on the right ("very beneficial"), with gray for "neutral" in the middle.

We can group this graph into three sections. Firstly, there are those aspects which mostly attracted negative answers, that is, MLSD is felt to be problematic for these aspects. This includes the lower four. 63% of respondents felt that the understandability of a system suffers when using multiple languages, and 53% see problems when changing code later on. About 50% of respondents see problems related to build management and developer effort.

Second, there are aspects where there is no clear trend towards benefits or problems. This includes the middle four aspects, which also have a high amount of votes for *neutral*. Thus, respondents do not agree on the effect on the initial implementation of the system,

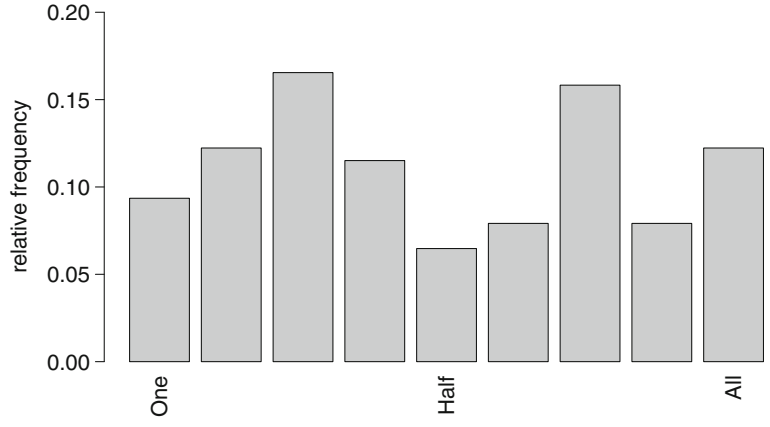

Fig. 7 Ratio of developers to languages: How many languages of the project did one developer work with?

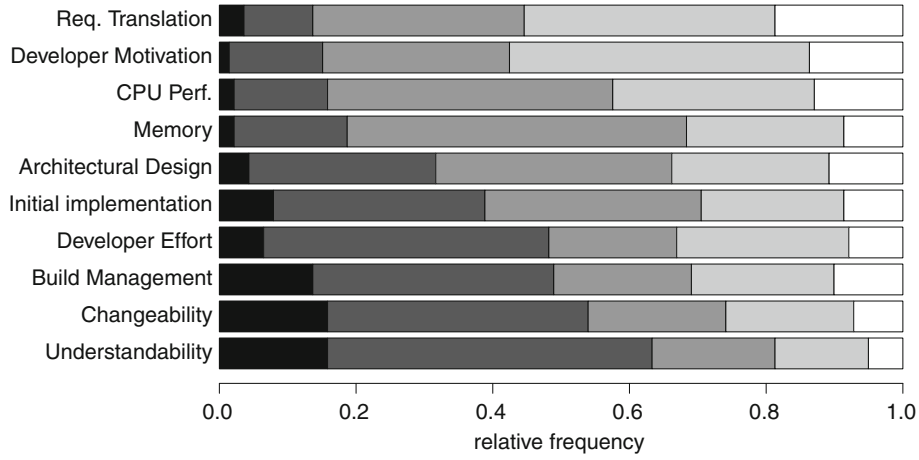

Fig. 8 Benefits and problems of multi-language software development. The bars are coded as follows (from left to right): *black*= very detrimental; *dark gray*= rather detrimental; *gray*= neutral; *light gray*= rather beneficial; *white*= very beneficial

the design of the architecture, as well as the technical factors of memory consumption and CPU performance.

Finally, the top two aspects show a clear trend towards a beneficial outcome. 57% of respondents feel that using multiple languages has a motivational effect on developers; 55% see a positive effect on the translation of requirements to code.

A rather high number of 29 of the 139 respondents added their own comments to this question. An interesting trend amongst these comments was the question of *how many people in the team are able to speak the required languages*, which is related to though not directly the same as the language *use* ratio we asked about in the previous question. One developer wrote that collective code-ownership and reviews become more difficult if only a part of the team is able to understand the code. Another pointed out that parallelizing tasks and thus utilizing all developers is not possible if only parts of the team speaks a certain language. Also, transferring knowledge in the team, and having surrogates for developers, was pointed out to become more difficult, as well as finding personnel with the required qualifications.

Several people also pointed out that the question of understandability depends on developer training. Several commented that the use of MLSD requires initial training for developers which takes time, but leads to later benefits due to the ability to use and understand additional languages. However, some cautioned that the problem of missing qualifications might crop up again when people leave the project or company, and might also be a problem in maintenance and support if different personnel is employed.

The last question in section one was about the trend towards or away from the use of multiple languages. Here, there is a clear trend, as shown in Table 2: 92% of respondents

Table 2 Past and future of multi-language software development: count of respondents and percentages are per question (row)

	Less	Same	More
Past	88 (63%)	40 (29%)	11 (8%)
Future	8 (6%)	61 (44%)	70 (50%)

feel that there were less (63%) or the same (29%) amount of languages in the past, whereas 94% feel that there will be the same (44%) amount or more (40%) in the future. Thus, there is the feeling that there is a trend towards more languages.

3.3 Section 2: cross-language linking

The second section of this questionnaire is about cross-language links, that is, connections between the code of artifacts written in different languages. The first three questions in this section are about the links themselves; respondents were asked to identify the language pairs in their last project which were connected by cross-language links. We separated these questions between GPL/DSL links (which we assumed to be most common) as well as GPL/GPL and DSL/DSL links.

A total of 130 respondents answered 'yes' to one of these three questions; that is, 9 out of 139 did not encounter any cross-language links in their last completed project.

The first question in this section is about links between general-purpose languages and domain-specific languages. Five language pairings occurred more than 30 times (20% of responses); these are shown in Table 3 (first section). All other pairs occurred only in a maximum of 5 responses each.

The second and third question were about GPL/GPL and DSL/DSL links. Here, only the respective first pairs (Java/JavaScript with 27 projects, HTML/CSS with 39 projects) occurred in any sizeable amount.

In total across these three questions, respondents reported a median number of 3 link pairs per project, with 50% of values between 2 and 5, and most (inner 80%) between 1 and 7. A total of 152 distinct link pairs were mentioned: 94 distinct pairs of GPL/DSL combinations, 32 of GPL/GPL combinations, and 26 DSL/DSL combinations.

The last three questions in this section are about possible problems occurring with cross-language links. All three questions were only asked if respondents answered yes to one of the three previous questions; that is, if cross-language links existed in their projects.

Question 10 asked respondents if they encountered any problems with cross-language links, and if so, of which type. Of the 130 participants answering this question, a total of 10 respondents (8%) reported no problems. The other 92% reported at least one problem. Figure 9 shows the distribution of problems (note that the selection was multiple choice).

The problems which occurred most with 80 respondents (62%) are bugs or issues resulting from changing cross-language identifiers. 60 respondents (46%) admitted to refraining

Table 3 Top linked language combinations

GPL/DSL				
Java/XML	Java/SQL	JS/HTML	Java/HTML	Java/.prop
44	38	33	32	31
GPL/GPL				
Java/JS	Java/Scala	Java/Groovy	Java/C	Java/Py
27	5	4	3	3
DSL/DSL				
HTML/CSS	XML/.prop	HTML/.prop.	HTML/XML	XML/SQL
39	9	5	5	5

JS = JavaScript; .prop = .properties; Py = Python

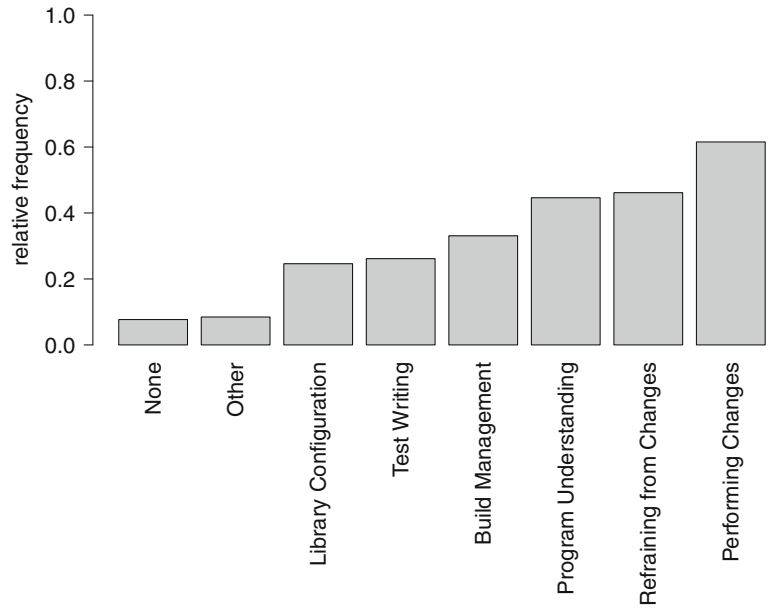

Fig. 9 Which problems did occur? (multiple choice)

from changing cross-language identifiers at all for fear of breaking the system. On third place, 58 participants (45%) found that program understandability suffered due to the existence of cross-language links.

The other three possible problems provided were issues in managing the build due to cross-language links, problems when writing test cases, or difficulties in configuring libraries; these problems were seen by 43, 34, and 32 respondents (33, 26 and 25%), respectively.

In the "other" category, additional problems where mentioned, namely an increased effort in debugging, difficulties in ensuring backwards compatibility in DSL code, the problem of using obsolete options or missing new ones in configuration files or translation files, and the increased effort for communicating such changes to colleagues. It was also pointed out that good tool support can solve (some of) these problems.

Question 11 asked again about problems with cross-language identifiers; however, this time we were interested in the frequency of occurrences in each software engineering activity. This question was only asked if respondents indicated in question 10 that problems did occur, which was the case for 120 of our participants. The results are shown in Fig. 10.

The figure shows the seven suggested activities on the y axis; the answers of respondents are shown as bars on the x axis. For each activity, respondents could select (from left to right) "never" (black), "rarely" (dark gray), "from time to time" (gray), "frequently", (light gray), and "all the time" (white). The figure shows the activities roughly ordered from project start (bottom) to project end (top).

In the first programming activity — initial implementation of the system — about 40% of respondents saw problems more than rarely. The bulk of issues was seen in the next two activities, which are about changes to the system: changing the code to implement new requirements, and refactoring. In the first, 78% of respondents saw problems more

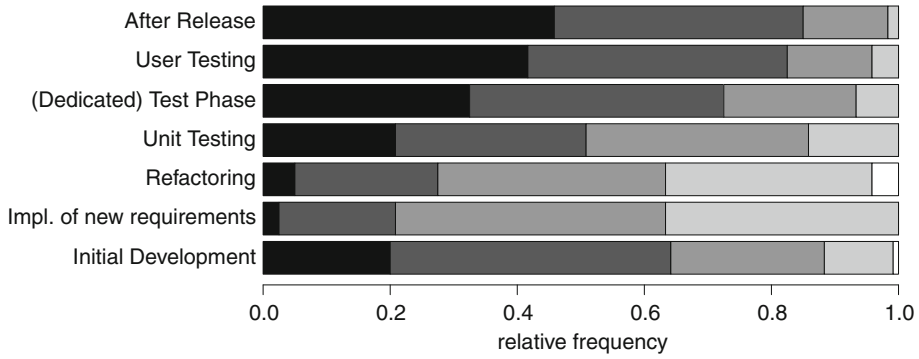

Fig. 10 *When* did problems occur, and how often? The bars are coded as follows (from left to right): *black*= did not occur; *dark gray*= occurred rarely; *gray*= from time to time; *light gray*= frequently; *white*= all the time

than rarely, in the second, 71%. Problems were still encountered, as they should, in unit testing by developers, where 49% of respondents saw problems more than rarely.

There is a drop in problem occurrence in the last three activities. In the first — dedicated testing phases performed by testing personnel — about 26% saw problems more than rarely, a figure which drops to 17% for user testing and 14% for issues that occurred after the release of the software.

The last question in this section was about measures taken to prevent or at least quickly find problems with cross-language links. This question was only asked to respondents who reported cross-language links in their project, that is, 130 of them. The results are shown in Fig. 11. Multiple answers could be selected.

A total of 12 respondents (9%) reported that no particular measures were taken. A sizeable number of respondents reported avoidance of the use of multiple programming

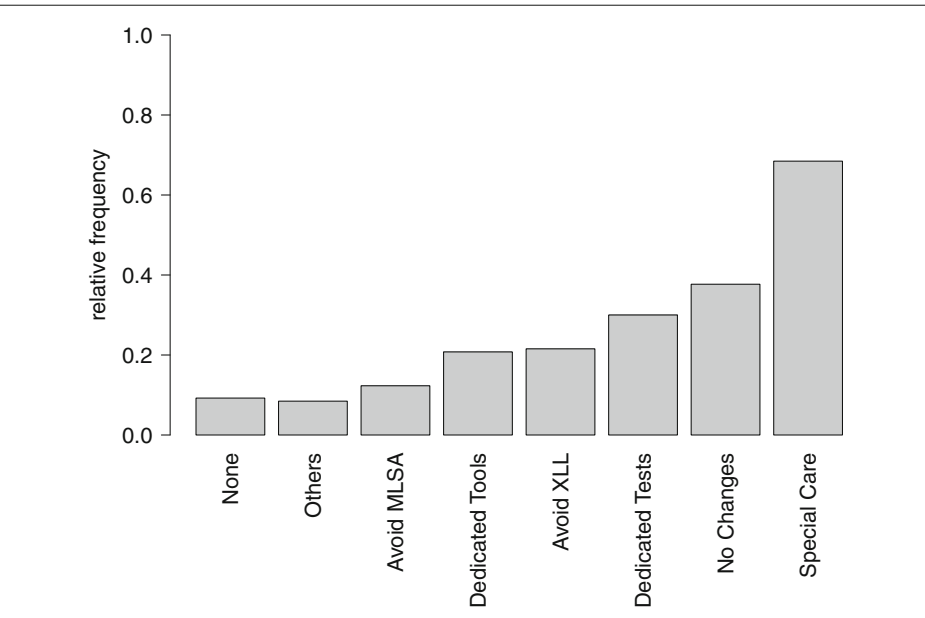

Fig. 11 Which measures were taken to address problems in cross-language linking? (multiple choice)

languages (16, 12%), avoidance of cross-language links (28, 22%) or avoidance of changing identifiers once they are in place (49, 38%). 89 respondents (68%) reported that special care was taken when changing cross-language identifiers.

Dedicated tools for handling cross-language identifiers were used by 27 respondents (21%), and specialized test cases for detecting cross-language issues were used by 39 respondents (30%).

In the "other" category, four people reported using code generation to prevent problems, that is, automatically generating at least parts of the cross-language links to ensure consistency. Two other measures mentioned were a) using clear coding conventions for cross-language identifiers, and b) the use of a central database for cross-linking identifiers which all languages had to refer to. Two respondents advocated the use of proper tool support to solve most of the issues.

3.4 Section 3: tool support

The final section of the questionnaire (as regards to content) is about tool support for cross-language linking. Question 13 asks which of four suggested functions for supporting cross-language links were present in the tools used in the respondent's last project. The result is shown in Fig. 12.

A total of 55 respondents (nearly 40%) reported that the tools they used had no cross-language specific functions whatsoever, which leaves about 60% of the respondents in which one or the other function was available. The most available function reported was highlighting of identifiers in the code (57 respondents, 41%), followed by rename functionality (45 respondents, 32%), navigation (i.e. "jumping" between occurrences) (38 respondents, 27%) and the marking of errors in cross-language links with the lowest number of respondents (35, 25%).

Nine respondents (6%) mentioned other tool support; three of these were again code generation as above; two mentioned custom implementations for checking code.

The last question (14) is again related to the overall experience of developers. We asked about the importance of tools for cross-language linking in general as well as individual tool functions. The result is shown in Fig. 13.

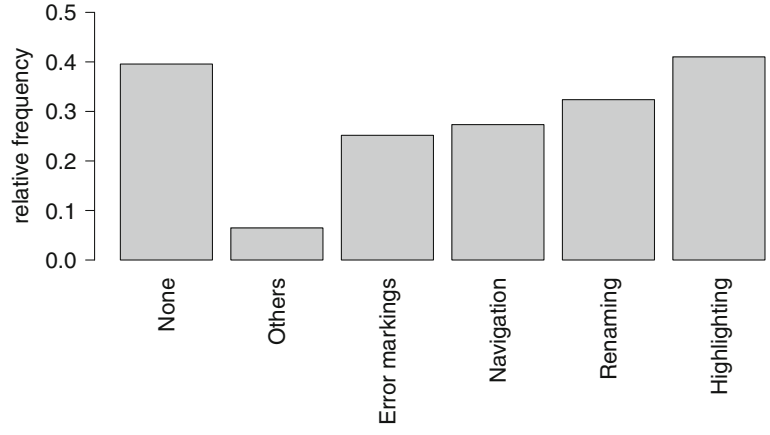

Fig. 12 Which tool functions were available in the tools used? (multiple choice)

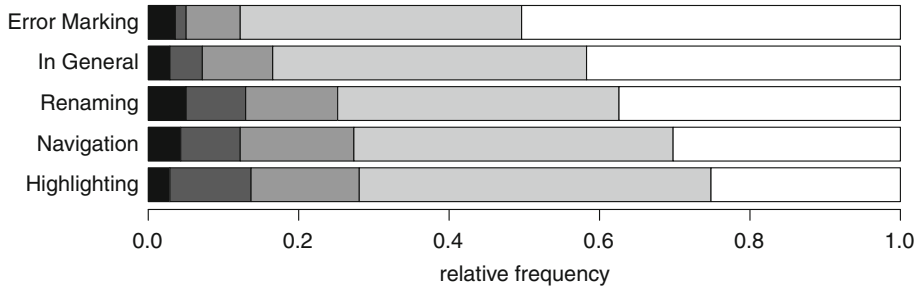

Fig. 13 Is tool support considered useful? The bars are coded as follows (from left to right): *black*= very detrimental; *dark gray*: rather detrimental; *gray*= neutral; *light gray*= rather beneficial; *white*= very beneficial

For each function and the general view, respondents could vote for (from left to right) "(very unimportant" (black), "rather unimportant" (dark gray), "neutral" (gray), "rather important" (light gray), and "very important" (white). For each function, over 70% of respondents reported tool support as important; in the general case, 82% of respondents found tool support to be of importance. The order of importance in functionality shows only slight deviations; highlighting is considered least important; renaming most important.

4 Discussion

We now discuss the study results shown in the previous section with regard to our research questions, thus explaining where and how this study has improved our knowledge of multi-language software development, cross-language linking, and accompanying tool support. We also indicate points where we feel practical efforts or future research would be beneficial.

Before we begin with the actual results, we discuss the metadata, or results from participant selection and demographic parts of the survey. As stated in the methods section, our selection was based on a form of snowballing and we thus refrain from generalizations. Based on the data we have gathered, however, it seems that the sample did include both a) knowledgeable respondents and b) a quite diverse set of participants.

First, with an average of 8 years of experience and 88% of participants indicating that they their responsibilities includes programming, we believe that the recipients indeed had knowledge about the questions posed in this questionnaire and the results are thus to be considered meaningful.

Second, the results show that many different levels of experience, team sizes, and project lengths are found in the sample. The years of experience range between from below 1 to 36 with 50% lying in between 5 and 15 years (median 8); the team size ranges between 1 and 170 with 50% between 4 and 18.5 developers (median 7), and the length of the projects range from 1 month to 5 years with 50% between 6.5 and 24 months (median 12). Furthermore, recipients worked in at least 23 different companies. This indicates that we achieved a good spread of people and projects which makes this sample worth investigating.

When we look at the types of software developed, we see a more uniform picture. 66% of the reported projects were either web applications or client/server applications. Server-only applications followed behind with 16% of responses, with all others below 9%.

Thus, the results of this survey should be interpreted with web, client/server, and server applications in mind — not, in particular, with desktop or mobile applications nor with embedded or operating systems.

The following three subsections discuss the answers to each of the three areas listed in the introduction, with two research questions each. We discuss threats to validity in Section 4.4.

4.1 On multi-language software development

In the following, we discuss the answers to research questions RQ1 and RQ2.

4.1.1 RQ1

Our first research question was *How prevalent is multi-language programming and which languages are used? How many languages did developers work with?*

Firstly, projects were reported to have an average of 7 languages (lower bound, since we counted DSL types instead of DSLs) with 50% of projects having between 6 and 8 languages. So, we conclude that multi-language programming is indeed prevalent.

We can also compare this figure with the number of languages in open source projects from a previous study (Mayer and Bauer 2015). Here, languages were automatically retrieved from source files with a median of 4 and 50% of projects having between 2 and 7 languages. It is thus interesting to see that more languages were reported in the current study, even if this count is a lower bound due to our asking for DSL *types*. This which may be due to various factors.

First, the current data is from the memory of participants, whereas the other is extracted from code; however, we would have expected a result in the other direction here (i.e. less reported languages from memory). The more likely explanation is that the data set in the previous study did include both a) unfinished projects and b) several smaller "toy" projects which reduced the number of languages. It would be interesting to compare the current results with a survey of open-source personnel.

The languages used mostly in our sample set were *Java and JavaScript* by a wide margin. This means in turn that the results of this study should be mostly interpreted with regard to projects with these languages, since we do not know the effect of the use of different general purpose languages on the questions asked.

With regard to DSLs, we only asked for *types*. Here, it has become abundantly clear that nearly all projects used the top five categories of DSLs, namely languages for the user interface, (shell) scripting, building, querying, and configuring. It will be interesting in the future to look into these categories in more detail and separate out the individual languages in use.

Only 12 respondents reported having created their own languages. Thus, although creating custom DSLs has been an active research topic for some time now (Fowler 2011), this was not a common phenomenon in our sample of industrial respondents.

It is important to note here that about 66% of respondents indicated that their last project was either a web application (36%) or a client/server application (30%). This might also indicate why the number of languages was higher than in our previous study, as the use of DSLs in those types of systems is very common.

Finally, there is no clear answer with regard to the question of the use of a project's languages by developers. Figure 7 has shown that basically all possible options from "just

one language" to "all of the languages" were selected. In general, there seem to be different philosophies at work regarding the allocation of work to developers or, more general, developer training. As we have seen in the comments to the question on benefits and problems, several recipients indicated problems precisely when not all team members were able to *understand* all languages, which is related to but not exactly the same question we asked here (which was about *writing code* in these languages). This seems like a natural step to follow up in future research.

To summarize our answer to RQ1: Multi-language programming is indeed common with an average of seven languages. Our sample set included mostly Java/JavaScript projects. Most projects used DSLs from the UI, configuration, shell scripting, querying, and building domains. Only 9% of respondents used custom languages. The developer-to-language ratio ranged freely from one language per developer to developers writing code in all present languages.

4.1.2 RQ2

Our second research question was *What are the benefits and problems developers encountered in the use of multiple languages? Do developers feel that multi-language programming has/will increase or decrease over time?*

Looking at the results from question five, respondents saw a benefit of the use of multi-language development in two areas. The first is a technical one, namely the *translation of requirements into code*. Thus, developers agree that certain languages are better suited to encode specific requirements than others. The second is related to a human issue, namely developer *motivation*, where a high number of respondents saw a benefit of using multiple languages.

On the other hand, a rather high number of respondents saw problems for the *understandability* of the system due to the use of multiple languages. This seems to conflict with the requirement translation answer above: Why do developers see multiple languages as beneficial for encoding requirements, but problematic for code understanding? We believe that the question of requirement translation was mostly interpreted regarding *single languages*, not the *combination of languages*. A single language can indeed be better suited to encode a problem; however, problems with understandability crop up when code is subsequently *combined* with others. This is an interesting discrepancy and should be followed up with further research.

The second area marked out to be problematic is the *changeability* of the system, that is, performing later changes in the presence of the use of multiple languages — developers in our sample mostly agree that multi-language software development leads to challenges when changing code later on.

Two additional areas are seen as problematic, which are the management of the build (which must take into account more languages and their respective tools) as well as the required effort for developers.

Finally, there is no clear trend to be seen with regard to the architectural design and the initial (first) implementation of the system, nor with regard to the two technical issues of memory consumption and CPU performance. We assume that these questions could not be answered on this level of abstraction, which would also explain the large set of neutral responses. Thus, we should follow up here with splitting these questions up with regard to individual languages or language types.

To summarize, the top two areas seen as problematic are related to understandability and changeability of the system. Both are important areas for system maintenance. Here, it seems prudent to follow up either with practical support in the form of tools, or with further research on techniques to improve these two areas *by design*.

The second part of RQ2 is about trends in multi-language programming. Here, respondents mostly agree that there were less languages in the past, and there will be more languages in the future, which is probably as expected and shows the need for further addressing multi-language software development.

To summarize our answer to RQ2: MLSD is seen as beneficial for developer motivation and for translating requirements to code, and seen as problematic for program understanding, changes to the system, build management, and developer effort. For architectural design, initial implementation, memory consumption, and CPU performance there is no clear trend. Developers agree that less languages were used in the past and more will be used in the future.

4.2 On cross-language linking

In the following, we discuss the answers to research questions RQ3 and RQ4.

4.2.1 RQ3

Our third research question was *In how many and which combinations of languages did developers encounter cross-language links?*. This question is interesting since answers to this question are difficult to extract in a generic fashion from source code (compared to, for example, the language counts of RQ1), since the language pairs and the linking mechanisms must be known before a link detection mechanism can be written, and there is a large number of possible link pairs and frameworks.

The results show that most links, as expected, occur between general-purpose languages and domain-specific languages. Since we already saw that Java and JavaScript are the top languages used in this sample, it is unsurprising to see them employed here as well, with languages from three of the five main DSL types as link targets (XML and .properties for configuration, HTML for UI, and SQL for querying) as shown in Table 3. As the table only shows the top 5, it is again unsurprising to see the "usual suspects" in GPL/GPL and DSL/DSL linking as well.

As we have mentioned above, the three questions on linking used free text only and were thus very tiresome to answer. We thus expect that many respondents only entered what immediately came to mind, and therefore the total numbers (3 link pairs per respondent) are probably below the actually occurring link pairs. 9 out of 139 respondents indicated that there were no cross-language links which seems surprising and might also be due to this issue. We took the reported language pairs at face value and did not take discrepancies with questions 1 and 2 into account.

Another issue we found here is that several respondents *wrongly attributed languages* to either general-purpose or domain-specific although these terms were explained in the survey. While this was easily corrected afterwards, it suggests that this distinction is not as well-known or clear-cut as we expected.

It is worth noting at this point that the questionnaire did not ask participants to separate between generated and non-generated code. Some code which includes cross-language links might in fact be generated (either in a GPL or in a DSL). This usually means that

the code is only read, but not (manually) changed. Still, developers need to be able to understand it.

With the answers to these questions we have gained information on link pairs which, as mentioned above, is hard to extract automatically. The full list of 152 distinct link pairs is available on our web site. We thus have a starting point for further research including writing tool support. For future surveys, we recommend using this information to create multiple-choice answers for these questions as well.

To summarize our answer to RQ3: Developers encountered cross-language links in a total of 152 distinct language pairs with an average of 3 link pairs per project. The most common combinations were GPL/DSL links between Java and XML, SQL, HTML, and .properties, as well as JavaScript and HTML.

4.2.2 RQ4

Our fourth research question was *Did problems with cross-language linking occur? If so, which, when, how frequent, and what was done to alleviate these problems?*. This question is at the heart of this study.

The first subquestion here is about whether problems with cross-language links occurred at all. Only about 8% of respondents reported no issues, so we can clearly state here that the overwhelming majority of respondents did encounter at least some issues with cross-language links.

Of the suggested issues with cross-language links, the most-selected one were problems as a result of changing cross-language identifiers (with 61% of respondents having had this problem in the last project). Another 46% stated that developers refrained from changing cross-language links for fear of breaking code. We added this answer based on previous experiences with industrial development without expecting that many developers would admit to having this issue. That they indeed did so is evidence, in our opinion, that cross-language links as they are now are indeed seen as being hard to keep track of.

While not changing identifiers prevents problems in the short run, it will lead to problems with understandability later on since necessary changes — such as renaming identifiers whose function has changed — are not carried out any longer. Indeed, the issue selected third most often (by 44%) is problems with understanding or explaining how the system worked due to cross-language links.

We conclude that cross-language links are seen as hard to keep track of and thus difficult to understand and communicate.

The other three suggestions of problems — increased difficulties in build management, test writing, and configuration of required libraries or frameworks — were selected by 30 to 40 percent of recipients. They are thus of concern as well, but less so than the problems discussed above.

In question 11, we asked respondents *when* and how frequently problems with cross-language links occurred during the development of their systems. The two activities most affected were those in which the system was *changed*, either to implement new functionality or when refactoring. A smaller additional amount of problems were detected (as should be) during unit testing.

Part of the aim of this question was to test which activities would likely profit from (tool) support and whether cross-language linking problems occur late in development,

making them harder to fix. Thus, the result is encouraging: Few respondents indicated that problems with cross-language links made it to user testing or were still present after release; instead, problems mostly occurred during programming when *changing code*. We conclude that future efforts to aid developers should be focused on these activities.

The last question here is about measures taken to alleviate problems with cross-language linking. It is interesting to see that despite 92% of recipients reporting problems with such links, only 20% used dedicated support tools and only 30% used dedicated tests to find such errors. About 9% reported that no measures were taken at all. It is unclear whether this is due to the fact that such tools simply do not exist for the relevant cross-language links, or whether they were not used for other reasons. This should be followed up in the future.

We also provided suggestions for "soft measures" as part of this question: 68% of respondents indicated that they took "special care" when changing cross-language identifiers. 12% reported avoidance of the use of multiple programming languages, and 21% reported avoidance of cross-language identifiers in general. 37% indicated that identifiers were not changed to avoid issues.

This again indicates that it seems difficult to feel comfortable with the presence of cross-language links and they are thus avoided when possible; if present, they are handled with the utmost care. This situation clearly needs additional attention both in practical efforts and in research.

To summarize our answer to RQ4: Problems with cross-language linking were reported by 92% of respondents. Most problems were related to changing cross-language identifiers and to program understanding, which suggests that cross-language links are seen as fragile and difficult to understand and communicate. These issues occurred mostly during activities in which code was changed by developers. Only about 20-30% of respondents used concrete measures against cross-language linking problems in the form of tools or test cases; many respondents indicated that they tried to avoid multiple languages, cross-language linking, or changing identifiers as far as possible.

4.3 On tool support

In the following, we discuss the answers to research questions RQ5 and RQ6.

4.3.1 RQ5

Our fifth research question was *Was tool support available for dealing with cross-language identifiers? If so, which functions were available?*.

55 (nearly 40%) of participants indicated that *no tool support* at all was used in their last completed project. Thus, tool support was in fact in use, but less so than we expected. In particular, the error marking functionality, which is the one directly relevant to problems — was selected by the lowest amount of recipients (35 recipients, 25%), which mirrors the result in question 12 (where 27 respondents (20%) reported using dedicated tools for error detection).

It is important to note here that we simply asked whether the functionality was available in the project. Whether this means that tools *do not exist at all or were simply not used* is unclear. This should be investigated in the future; we suggest asking about concrete tools in combination with cross-linked language pairs.

To summarize our answer to RQ5: Tool support was available to about 60% of respondents. The functionality most available was highlighting (41%), followed by renaming, navigation, and finally error marking (25%).

4.3.2 RQ6

Our sixth and final research question was *Is tool support considered important, and if so, which functions for handling cross-language identifiers are most important?*.

In this final question, recipients agreed that having more tool support would be beneficial. A total of 82% of recipients indicated that they see tool support in general as "very" or "rather" important. Thus, it seems clear that there is a need for further support. There is little difference between the individual functions in rating. The most important functionality to recipients is, as expected, support for error marking (87%). The least important was support for highlighting (67%), although it should be noted here that highlighting was the functionality most available to developers and thus may have received less votes.

To summarize our answer to RQ6: Respondents universally agree on the benefits of tool support for cross-language linking. The most benefits (87%) are expected from functionality to mark errors in cross-language links.

4.3.3 Closing remarks

In the previous subsections we have answered our research questions in great detail. We have found that MLSD and the use of cross-language linking are indeed prevalent, as are related problems. We see three areas in which we might improve the state of the art: Better tool support, easier cross-language linking *by design*, and more focus on the ability of developer teams to speak all languages in use in a system.

We attribute problems with understandability and changeability to the fragile and implicit nature of cross-language links as they exist today — it seems hard to keep track of such links. Two remedies suggest themselves for this issue. The first is tool support, which has already been suggested as part of this survey and has been met with universal agreement by developers: Better tool support for tracking and changing cross-language links may indeed alleviate many problems associated with these links, including the fear of developers to change code due to unknown cross-language effects and thus reestablishing trust in the code base.

However, better tool support can also be seen as only handling the *symptoms* of the problem: If cross-language linking mechanisms were more robust in the first place, we would not have as great a need of tool support as we do now. Thus, secondly, we should investigate creating maintainable and understandable cross-language links *by design*. How specific cross-language linking mechanisms could be improved is a matter of future research. In the comments, several respondents mentioned the use of code generation tools with the benefit of cross-language linking information being stored in only one place. One respondent explicitly mentioned using a central database for this information which is a form of explicit interface specification. A first step forward might thus be made by drawing on the experience with interface specifications in general, that is, using more explicit and accountable links which are stored in well-known places.

Third, the qualitative data from the comments of developers on various questions show one potentially underinvestigated area, which is the knowledge of team members about

the languages used in the project. This problem is related to but not the same as our question on the developer/language ratio, which was more about *changing*, not only *understanding* code. Several respondents have commented that various problems occur if not all team members speak all languages, not only for the code quality, but also for organizational issues such as finding replacements during a vacation or when developers leave a company. We suggest that this angle of the problem be followed up in the future as well.

4.4 Threats to validity

As with any empirical study, there may be threats to the validity and trustworthiness of our work (Kitchenham and Pfleeger 2002b). Since this is the first survey in this area we cannot compare to existing results or instruments. We thus discuss content validity as perceived by the authors, pre-testers, and the survey respondents (in the form of free text comments).

A general issue with questionnaires is the risk of participants not understanding the questions or (pre-made) answers. We have taken care to adapt the terms to the intended target audience, i.e. professional developers, not researchers. We have refrained from using too technical terms by replacing them with simpler ones; where this was not possible we have explained the terms (such as "cross-language link") within the questionnaire: each section of the questionnaire was prefixed with a page of explanations. We have also provided examples where appropriate, for example, for the distinction between GPL and DSL. Furthermore, the questionnaire was run through a pre-test with five participants. We thus believe it unlikely that there were serious misunderstanding of the questions.

The question on benefits and problems of MLSD in this questionnaire was kept generic, i.e. participants were asked to answer this question regardless of concrete languages or language types. This was done on purpose as it was our aim to find out if there are trends in opinion when considering the MLSD phenomenon as a whole; however, we might have attracted more insights had we asked this question multiple times for individual languages or types. A "neutral" option was provided to enable participants to refrain from having to choose, an option which was made use of in several instances as reported. While we believe that the results to this question are a meaningful first step, we recommend that future studies split this question to individual languages and types.

Most questions in this questionnaire included multiple-choice answers, i.e. we provided prefixed answers instead of open ones. The main benefits of this approach are a) it is easier for respondents to answer, and b) the results are easier to analyze. However, this may lead to a bias since participants were able to select pre-fabricated opinions instead of having to provide their own ones. All options provided in the questionnaire were discussed among the authors and, where possible, with other developers, and thus stem from the experience of the authors and other software developers in the development of multi-language software.

It is worth noting at this point that open questions are problematic as well since having to provide free text increases the likelihood of participant dropouts. Furthermore, analysis of such data would again require categorization which is then performed by the researcher, not the respondent. To address this issue, we provided both suggestions and an additional free text field in all of these questions such that developers could add their own opinions. We investigated these qualitative answers manually, but mostly did not find a clustering of non-listed issues except in two cases: Several respondents mentioned

additional problems if not all team members are able to *understand* all languages; and several respondents indicated the use of *code generation*, that is, using a single specification for cross-language links and using this specification to prevent problems. In a follow-up study, these two topics should thus be investigated further.

We finally come to the question of the ability to generalize. The selection of participants for this study was done in a snowball fashion, the reasons for which have to do with our inability to randomly select participants due to the inaccessibility of the population and questions regarding unsolicited e-mail. We have thus, in this report, refrained from using inferential statistics and have only reported our results on this sample. As we have indicated in the results section, we feel that the sample has a good diversity nevertheless and as such we feel that the results offer interesting insights into industrial development. Also, parts of this study are of an explorative nature, and we feel that our results directly suggest further research as indicated in the discussion section.

Even when reporting on the sample only, there is one possible bias which comes from the threat that multiple respondents in our survey may have reported on the same project. However, most companies in our sample offer customized development to clients and thus have many individual projects running at the same time. This problem also only affects the questions on the last completed project, not the questions on developers opinions. The free text answers did not show any patterns to confirm this concern, and we assume that the overall diversity was high enough to counter this issue.

5 Related work

To our knowledge, this is the first survey on either of the topics of multi-language software development, cross-language linking and accompanying tool support on actual developers (industrial or otherwise), with one exception (Pfeiffer and Wasowski (2015), see below). However, there is existing non-survey work in all three areas related to this study which we discuss in the following.

5.1 Multi-language software development

To our mind, research on Multi-Language Software Development (MLSD) deals specifically with the combined occurrence and use of programming languages; not with individual programming languages themselves, but neither with cross-language links specifically. We can find references to this practice (sometimes also called *mixed-language programming* or *polyglot programming*) as far back as 1984 (Einarsson and Gentleman 1984); however, most work is more recent.

Directly relevant to this study are three recent data mining studies on open source software (Mayer and Bauer 2015; Tomassetti and Torchiano 2014; Delorey et al. 2007). These studies focus on co-occurence of languages. In all of these studies, open source projects have been analyzed programmatically. In the first two studies, GitHub was used as the provider; in the third, SourceForge. These first two studies found a median of 4 resp. 5 languages per project, which is lower than the one reported here (which is 7), despite the fact that the number we reported here is a lower bound since we did not ask for DSLs but for DSL *types*.

The number of languages in use in industry thus seems to be higher. We believe that this is due to the fact that many projects on GitHub are a) not production-ready and b) are smaller, personal projects.

The SourceForge study Delorey et al. (2007) does not explicitly list a median number of all languages, but mentions the use of two to three GPLs per project, which is similar to what we see here.

Another interesting aspect of MLSD is the evolution of projects over time; this topic has been investigated by Arbuckle (2011). In this case study of *git*, it was found that code in scripting languages was not, as expected, replaced by general-purpose language code over time, but that instead the development of all languages proceeded in sync.

There is no related work which address our research question two, i.e. whether MLSD is perceived as beneficial or problematic by developers.

5.2 Cross-language linking

The area of cross-language linking is about the actual interaction points between languages, that is explicit or implicit interfaces which often use *named identifiers* for establishing links. There is no empirical evidence on the incidence of such links in software projects, which is mostly due to the fact that such links cannot be detected *in general*: one has to implement specific support for the links between each language pair and usually for the concrete linking mechanism (often a framework or library) as well.

Thus, our answer to research question three on cross-language link occurrence is, in its generality, new data on this subject.

The fact that cross-language links may be problematic has been taken for granted in much of the literature on cross-language links and tool support (see below), and several suggestions have been made to improve the situation.

Favre et al. (2012) and Lämmel and Varanovich (2014) have investigated the question of the linguistic architecture of projects with a specific focus on *modeling*, that is, design-time support. Such support may help alleviate the reported problems of understandability in such systems as we have seen in research question four.

There are also efforts regarding to *researching* cross-language links. Caracciolo et al. (2014) have created a workbench for running analyses on multi-language software systems; Sobernig and Zdun (2010) have addressed the topic of how to evaluate execution speed of cross-language method invocations in Java. Both are thus enablers for future efforts which may go into the directions of tool support or better linking mechanisms we have indicated.

We do not know of related work for our research question four, that is generic data on problems and measures taken with regard to cross-language links.

5.3 Tool support for cross-language links

Most research on cross-language linking is tightly coupled with the question of tool support. The first such work is probably that of Linos et al. from 1995 on a tool for re-engineering cross-language links Linos (1995). Since then, several publications have addressed cross-language links and tool support for a variety of language combinations.

In our questions on tool support, we have asked for various functions which may help in dealing with cross-language links. Such functionality has indeed been implemented. For example, Mayer and Schroeder (2014) describe a tool for generic support of cross-language linking in Eclipse which includes all of these functions. There are also

other tools which offer such functionality; examples are Pfeiffer R and Wasowski (2012b), Nguyen et al. (2012), and Tomassetti et al. (2014). Sakamoto et al. (2013) have created a tool for test coverage measurement which spans languages.

There is also more general work on tool support for cross-language linking. In 2012, Pfeiffer and Wasowski (2012a) performed an experiment to assess the usefulness of tool support for cross-language linking; their result that such support aids software development matches our result in research question six where developers offered much the same opinion.

The overall design space of such tool support is discussed by Pfeiffer and Wasowski (2015). This work also includes a survey of *language designers* in which designers were asked questions on the new languages they designed. The results of this study are complementary to our research question three in that developers indicated that their new languages in most cases interacted with others (84%). It also touches our research question five: designers indicated that they provided tools for checking these interactions (68%), but that their tools were not generic (8%); i.e., they were focused on the concrete languages. The survey also includes a question on how many languages were present in respondents' projects, which yielded a number of between 3 and 8.5 (compared to our median of 7 in RQ1).

There is no direct related work which addresses our research question five, that is whether such tool functionality is available and used in industry in general.

6 Conclusions and outlook

In this study we surveyed 139 industrial software engineers for their opinions on multi-language software development, cross-language linking, and accompanying tool support.

6.1 Summary

The participants of this study were all software engineering professionals with an average of 8 years of experience; most were responsible for programming (88%). The software development projects they reported on had an average of 7 team members and a duration of one year. The systems under development were mostly web and client/server applications (66%) or server only (16%).

Regarding multi-language software development in general, our respondents reported about 7 languages per project with Java and JavaScript as the main languages, and a ubiquitous use of DSLs from the UI, configuration, querying, shell scripting, and build management domains. 9% used custom languages. The developer-to-language ratio ranged freely between one language and all languages per developer. Multi-language software development is seen by respondents as beneficial for developer motivation and translation of requirements to code, but as problematic for understandability and changeability of the system. Participants agree that there will be more languages per project in the future.

Regarding cross-language linking and identifiers, developers encountered cross-language links in a total of 152 distinct language pairs with an average of 3 link pairs per project. 92% of respondents reported having encountered problems with cross-language links. Most problems were (again) related to changeability and understandability, and occurred during the programming phase of the project. Only about 20–30% of respondents reported actual measures against cross-language linking issues; many

indicated avoidance of multi-language development, cross-language linking, or changing cross-language identifiers.

Tool support for cross-language linking was available to about 60% of our respondents, with automated error detection available to only 25%. Respondents universally agree on the benefits of tool support, especially for error marking.

6.2 Conclusions and outlook

This paper has reported the results of a descriptive study with explorative elements. We believe that the results of this study can help us focus both practical efforts and further research in three areas.

First, we have seen that most respondents indicated having problems with cross-language links, mostly related to link changeability and understanding software during the programming phase of a software project, and that tool support seems both in short supply and perceived as beneficial. Thus, creating *tool support* in particular for error marking in cross-language links might offer developers a better safety net than they currently have. For program understanding, accountability of cross-language links through advanced querying mechanisms or visualization might be beneficial to avoid the tendency to not change code for fear of breaking the system.

Second, creating more tool support can also be seen as only addressing the *symptoms* of badly designed cross-language linking mechanisms. In fact, less fragile links (for example, through more explicitly defined and maintained interfaces) might lead to better changeability and understandability *by design*. An interesting question is whether there are already subgroups of link types which offer these attributes. If so, we can favor these types of links in future implementations. A follow-up survey with language- or link-specific questions might answer this question.

Last, several respondents remarked on various problems occurring if not all team members are able to understand all languages. This, in particular, applied to organizational issues such as replacing developers or parallelizing tasks. This suggests also that some problems may need to be tackled with better training and not (only) better technology.

Endnotes

[1] The full questionnaire as presented to respondents can be found on http://xll.pst.ifi.lmu.de/survey.html.

[2] http://soscisurvey.de/

[3] http://xll.pst.ifi.lmu.de/survey.html

[4] http://xll.pst.ifi.lmu.de/survey.html

Acknowledgements

The authors would like to thank all participants in the survey for their time and their answers without which this study would not have been possible. Furthermore, we would like to thank our colleagues for their efforts in recruiting participants for this study. Thanks also go out to SoSciSurvey for providing access to their excellent online survey tool, to Sonja Pointner from the Social Science department of LMU for answering our questions on questionnaire design, and to André Klima from the Statistical Consulting Unit StaBLab of LMU for his support regarding the statistical analysis.

Funding

The first author has been supported by the DFG project MA 794/9-1.

Availability of data and materials

The raw data accompanying this work is available on http://xllsrc.net/survey.

Authors' contributions
The first author (PM) is responsible for the overall idea behind this work as well as the main execution of the survey, data analysis, and dissemination effort. The second author (MK) has provided support in the creation of the survey questions as well as the online survey on the SoSciSurvey platform. The third author (MAL) has provided support with regard to the statistical analysis in this work. All authors read and approved the final manuscript.

Competing interests
There are no competing interests for any of the authors.

Publisher's Note

Author details
[1]Programming & Software Engineering Group, Ludwig-Maximilians-Universität München, München, Germany.
[2]Statistical Consulting Unit StaBLab, Ludwig-Maximilians-Universität München, München, Germany.

Received: 12 July 2016 Accepted: 9 March 2017
Published online: 19 April 2017

References

Arbuckle T (2011) Measuring multi-language software evolution: a case study. In: Cleve A, Robbes R (eds). Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, EVOL/IWPSE 2011, September 5-6. ACM, Szeged, Hungary. pp 91–95. doi:10.1145/2024445.2024461

Caracciolo A, Chis A, Spasojevic B, Lungu M (2014) Pangea: A workbench for statically analyzing multi-language software corpora. In: 14th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2014, September 28-29, 2014. IEEE Computer Society, Victoria. pp 71–76. doi:10.1109/SCAM.2014.39. http://dx.doi.org/10.1109/SCAM.2014.39

Delorey DP, Knutson CD, Giraud-Carrier C (2007) 2nd International Workshop on Public Data about Software Development *WoPDaSD '07*, Springer, Heidelberg, Limerick

Einarsson B, Gentleman WM (1984) Mixed language programming. Softw Pract Exper 14(4):383–392. doi:10.1002/spe.4380140410. http://dx.doi.org/10.1002/spe.4380140410

Favre J, Lämmel R, Varanovich A (2012) Modeling the linguistic architecture of software products. In: France RB, Kazmeier J, Breu R, Atkinson C (eds). Model driven engineering languages and systems. MODELS 2012. pp 151–167. doi:10.1007/978-3-642-33666-9_11. http://dx.doi.org/10.1007/978-3-642-33666-9_11

Fowler M (2011) Domain-Specific Languages. The Addison-Wesley signature series. Addison-Wesley Professional, Indianapolis. http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0321712943,00.html

Franzen A (2014) Antwortskalen in standardisierten befragungen. In: Baur N, Blasius J (eds). Handbuch Methoden der empirischen Sozialforschung. Springer Fachmedien, Wiesbaden. pp 701–711. doi:10.1007/978-3-531-18939-0_51. http://dx.doi.org/10.1007/978-3-531-18939-0_51

Kitchenham B, Pfleeger SL (2002a) Principles of survey research: part 5: populations and samples. ACM SIGSOFT Softw Eng Notes 27(5):17–20. doi:10.1145/571681.571686. http://doi.acm.org/10.1145/571681.571686

Kitchenham BA, Pfleeger SL (2002b) Principles of survey research: part 3 constructing a survey instrument. ACM SIGSOFT Softw Eng Notes 27(2):20–24. doi:10.1145/511152.511155. . http://doi.acm.org/10.1145/511152.511155

Lämmel R, Varanovich A (2014) Interpretation of linguistic architecture. In: Cabot J, Rubin J (eds). Modelling Foundations and Applications - 10th European Conference, ECMFA 2014, Held as Part of STAF 2014, York, UK, July 21-25 2014 vol 8569. Springer, Heidelberg. pp 67–82. doi:10.1007/978-3-319-09195-2_5. http://dx.doi.org/10.1007/978-3-319-09195-2_5

Linos PK (1995) Polycare: a tool for re-engineering multi-language program integrations. In: 1st IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '95), November 6-10, 1995. Fort Lauderdale. IEEE Computer Society. p 338. doi: 10.1109/ICECCS.1995.479355. http://dx.doi.org/10.1109/ICECCS.1995.479355

Mayer P, Bauer A (2015) An empirical analysis of the utilization of multiple programming languages in open source projects. In: Lv J, Zhang HJ, Babar MA (eds). Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, EASE 2015, April 27-29. ACM, Nanjing. pp 4:1–4:10. doi:10.1145/27458022745805. http://doi.acm.org/10.1145/2745802.2745805

Mayer P, Schroeder A (2014) Automated multi-language artifact binding and rename refactoring between java and dsls used by java frameworks. In: Jones R (ed). ECOOP 2014 - Object-Oriented Programming - 28th European Conference, Uppsala, Sweden, July 28 - August 1, 2014. Proceedings, Springer, Lecture Notes in Computer Science Vol. 8586. pp 437–462. doi:10.1007/978-3-662-44202-9_18. http://dx.doi.org/10.1007/978-3-662-44202-9_18

Nguyen HV, Nguyen HA, Nguyen TT, Nguyen TN (2012) Babelref: Detection and renaming tool for cross-language program entities in dynamic web applications. In: Glinz M, Murphy GC, Pezzè M (eds). 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012. IEEE, Zurich. pp 1391–1394. doi: 10.1109/ICSE.2012.6227240. http://dx.doi.org/10.1109/ICSE.2012.6227240

Pfeiffer R, Wasowski A (2012a) Cross-language support mechanisms significantly aid software development. In: France RB, Kazmeier J, Breu R, Atkinson C (eds). Model driven engineering languages and systems. MODELS 2012. pp 168–184. doi: 10.1007/978-3-642-33666-9_12. http://dx.doi.org/10.1007/978-3-642-33666-9_12

Pfeiffer R, Wasowski A (2012b) Texmo: A multi-language development environment. In: Vallecillo A, Tolvanen J, Kindler E, Störrle H, Kolovos DS (eds). Modelling Foundations and Applications - 8th European Conference, ECMFA 2012, Kgs. July 2-5, 2012. vol 7349. Proceedings, Lyngby, Springer, Lecture Notes in Computer Science. pp 178–193. doi:10.1007/978-3-642-31491-9_15. http://dx.doi.org/10.1007/978-3-642-31491-9_15

Pfeiffer, R, Wasowski A (2015) The design space of multi-language development environments. Softwa Syst Model 14(1):383–411. doi:10.1007/s10270-013-0376-y. http://dx.doi.org/10.1007/s10270-013-0376-y

Core Team (2015) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. Vienna, Austria, http://www.R-project.org/

Sakamoto K, Shimojo K, Takasawa R, Washizaki H, Fukazawa Y (2013) OCCF: A framework for developing test coverage measurement tools supporting multiple programming languages. In: Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013 March 18-22. Luxembourg. IEEE Computer Society, Luxembourg. pp 422–430. doi:10.1109/ICST.2013.59. http://dx.doi.org/10.1109/ICST.2013.59

Sobernig S, Zdun U (2010) Evaluating java runtime reflection for implementing cross-language method invocations. In: Krall A, Mössenböck H (eds). Proceedings of the 8th International Conference on Principles and Practice of Programming in Java, PPPJ 2010 September 15-17. ACM, Vienna. pp 139–147. doi: 10.1145/18527611852781. http://dx.doi.org/10.1145/18527611852781

Tomassetti F, Torchiano M (2014) An empirical assessment of polyglot-ism in github. In: Shepperd MJ, Hall T, Myrtveit I (eds). 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, May 13-14, 2014. ACM, London. pp 17:1–17:4. doi:10.1145/26012482601269. http://doi.acm.org/10.1145/2601248.2601269

Tomassetti F, Rizzo G, Troncy R (2014) Crosslanguagespotter: a library for detecting relations in polyglot frameworks. In: Chung C, Broder AZ, Shim K, Suel T (eds). 23rd International World Wide Web Conference, WWW '14, April 7-11, 2014. Companion Volume. ACM, Seoul. pp 583–586. doi:10.1145/2567948.2578036. http://doi.acm.org/10.1145/2567948.2578036

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B (2012) Experimentation in Software Engineering. Springer, Heidelberg. doi:10.1007/978-3-642-29044-2. http://dx.doi.org/10.1007/978-3-642-29044-2