What Programming Languages Do Developers Use? A Theory of Static vs Dynamic Language Choice

Aaron Pang, Craig Anslow, James Noble
School of Engineering and Computer Science
Victoria University of Wellington, New Zealand
Email: {pangaaro, craig, kjx}@ecs.vuw.ac.nz

***Abstract*—We know very little about why developers do what they do. Lab studies are all very well, but often their results (e.g. that static type systems make development faster) seem contradicted by practice (e.g. developers choosing JavaScript or Python rather than Java or C#). In this paper we build a first cut of a theory of why developers do what they do with a focus on the domain of static versus dynamic programming languages. We used a qualitative research method – Grounded Theory, to interview a number of developers (n=15) about their experience using static and dynamic languages, and constructed a Grounded Theory of their programming language choices.**

I. Introduction

With an increasingly number of programming languages, developers have a wider set of languages and tools to use. Static languages are generally considered to have inflexible code and logical structures, with changes only occurring if the developer makes them. While dynamic languages allow greater flexibility and are easier to learn. It can be difficult to select a language(s) for a given project. There is little research regarding why and how developers make the languages choices they do, and how these choices impact their work.

Over the last decade there has been an increase in the use of dynamic languages over static ones. According to a programming language survey in 2018 [1], three of the five most popular programming languages are dynamic. These are Python, JavaScript, and PHP which accounted for 39%, while the remaining two C# and Java accounted for 31%. In 2007 the top five were Java, PHP, C++, JavaScript, and C.

In this paper we investigate why and how developers choose to use dynamic languages over static languages in practice and vice versa. We created a *theory of static vs dynamic language choice* by interviewing developers (n=15) and using Grounded Theory [2], [3]. The theory discusses how three categories (attitudes, choices, and experience) influences how developers select languages for projects, the relationships between them and the factors within these categories. Attitudes describes the preconceptions and biases that developers may have in regard to static or dynamic languages, while choice is the thought process a developer undergoes when selecting a programming language, and experience reflects the past experiences that a developer has had with a language. The relationships between these categories was that attitudes informs choice, choice provides experience, and experience shapes attitudes.

II. Related Work

A number of papers about programming languages have conducted empirical studies, controlled experiments, surveys, interviews with developers, and analysis on repositories.

Paulson [4] discusses the increase in developers using dynamic languages rather than static ones. Paulson claims that developers wish to shed unneeded complexity and outdated methodologies, and instead focus on approaches that make programming faster, simpler, and with reduced overhead.

Prechelt and Tichy [5] conducted a controlled experiment to assess the benefits of procedure argument type checking (n=34, with ANSI C and K&R C, type checked and non-type checked respectively). Their results indicated that type-checking increased productivity, reduced the number of interface defects, and reduced the time the defects remained throughout development. While Fischer and Hanenberg [6] compared the impact of dynamic (JavaScript) and static languages (TypeScript) and code completion within IDEs on developers. The results concluded that code completion had a small effect on programming speed, while there was a significant speed difference between TypeScript (in favour of) and JavaScript. A further study [7], compared Groovy and Java and found that Java was 50% faster due to less time spent on fixing type errors which reinforces findings from an earlier study [8]. Another study compared static and dynamic languages with a focus on development times and code quality using Purity (typed and dynamically typed versions). Results showed that the dynamic version was faster than the static version for both development time and error-fixing contradicting earlier results [9].

Pano et al. [10] interviewed developers to understand their choices of JavaScript frameworks. The theory that emerged was that framework libraries were incredibly important, frameworks should have precise documentation, cheaper frameworks were preferred, positive community relationships between developers and contributors provided trust, and that developers highly valued modular and reliable frameworks.

Meyerovich and Rabkin [11] conducted surveys to identify the factors that lead to programming languages being adopted by developers. Analysis of the surveys lead to the identification of four lines of inquiry. For the popularity and niches of languages, it was concluded that popularity falls off steeply and flatlines according to a power law, less popular languages

have a greater variation between niches and developers switch between languages mainly due to its domain and usage rather than particular language features. In terms of understanding individual decision making regarding projects, they found that existing code and expertise were the primary factors behind selecting a programming language for a project, with the availability of open-source libraries also having a part. For language acquisition, developers who had encountered certain languages while in education were more likely to learn similar languages faster while in the workforce. Developer sentiments about languages outside of projects were examined with developers tending to have their perceptions shaped by previous experience and education. Developers tended to place a greater emphasis on ease and flexibility rather than correctness, with many of those surveyed being pre-disposed to dynamic languages. They concluded that all of these factors are relevant to the adoption of programming languages.

Ray et al. [12] conducted a large study of GitHub projects (n=729, 17 of the most used languages and the top 50 projects for each of these), to see the impact static and dynamic languages as well as strong and weak typing have on the quality of software. Projects were split into different types and quality was analysed by counting, categorising, and identifying bugs. They concluded that static typing is better than dynamic typing and strong typing better than weak.

Prior research has not explained why developers use certain programming languages for work and personal projects and why there is an increase in the use of dynamic languages. However, the results of prior research will help inform our research as it analyses where dynamic development may be utilized rather than static development, as well as running counter to the belief that statically typed development is always faster, less error-prone, and easier to fix than dynamically typed development. The results will be in turn used in our study to identify potential avenues of questioning for data collection and analysis, allowing us to identify why developers use static or dynamic languages.

III. Methodology

We used the Grounded Theory (GT) method as it supports data collection via interviews and we were primarily concerned with the subjective knowledge and beliefs that developers hold, rather than technical ability [2], [3]. As our research focuses on people and the decisions that they make in regards to programming, GT is appropriate to study these behaviours and interactions particularly for software development projects as used elsewhere [13]–[20]. Human ethics approval was obtained. There are several stages to GT. Upon identifying a general research topic, the first step is the sampling stage, where potential participants are identified and a data collection method selected. We used interviews for data collection and transcribed each interview from audio recordings. Next is data analysis which uses a combination of open coding and selective coding. Coding is where key points within the data are collated from each interview and summarised into several words. These codes can be formed into concepts, which are patterns between groups of codes and then categories, which are concepts that have been grown to encompass other concepts. Amongst these categories a core category will emerge, which will become the primary focus of the study. Selective coding can then be used which only deals with the core category. Throughout the process is the memoing task, where ideas relating to codes and their relationships are recorded [21]. By recording all of these memos, it allows knowledge about what is emerging from the data and its analysis. One can then revisit the data collection phase and adjust their approach to specifically ask participants questions related to the core category. In order to create a theoretical outline, the memos are conceptually sorted and arranged to show the concept relationships. This theoretical outline will show how other categories are related to the earlier identified core category. Theoretical saturation is when data collection is completed and no new top-level categories are being generated.

TABLE I: A summary of the participants. Participant ID, Role Type based on developer role, Experience in number of years, Organization Type, Programming Languages experience in their main top two languages.

PID	Role	Exp	Organization	Languages
P1	Graduate	1	Government	Java, JavaScript
P2	Graduate	1	Finance	C#
P3	Graduate	1	Accounting	JavaScript, C#
P4	Graduate	1	Development	Java, Python
P5	PhD Student	4	Energy	Java, Coq
P6	PhD Student	4	Education	JavaScript, Python
P7	Intermediate	>5	Consultancy	C#, JavaScript
P8	PhD Student	1	Education	Python, C++
P9	Senior	>10	Self-Employed	Python
P10	Senior	40	Consultancy	Python
P11	Senior	10	Development	C++ , Objective C
P12	Senior	>10	Development	JavaScript, TypeScript
P13	Graduate	4	Development	Java, JavaScript
P14	Intermediate	>5	Development	Clojure, JavaScript
P15	Intermediate	>5	Development	Clojure, JavaScript

The first author conducted interviews with 15 participants, see Table 1. The interview schedule was updated after the first interview in order to provide a greater depth of questioning. The initial schedule only had broad sections, whereas the revised schedule had specific questions within each section indicating potential lines of questioning. By asking more open-ended questions, we were able to attain high-quality responses that contained more information. Once emergent information became apparent, questions in future interviews were modified in order to reflect these trends. From the interviews several concepts emerged from the open codes. The concepts have been formed by identifying groups of codes that have broad similarities (some codes were in multiple concepts) which helped to find the main theme of the research. Aggregating the codes helped to inform the factors that determine why developers make the choices they do regarding utilising static or dynamic languages. The first author identified the codes and to support reliability the others validated them to decide the concepts. Further details about the interview procedure, interview data, and coding results can be found elsewhere [22].

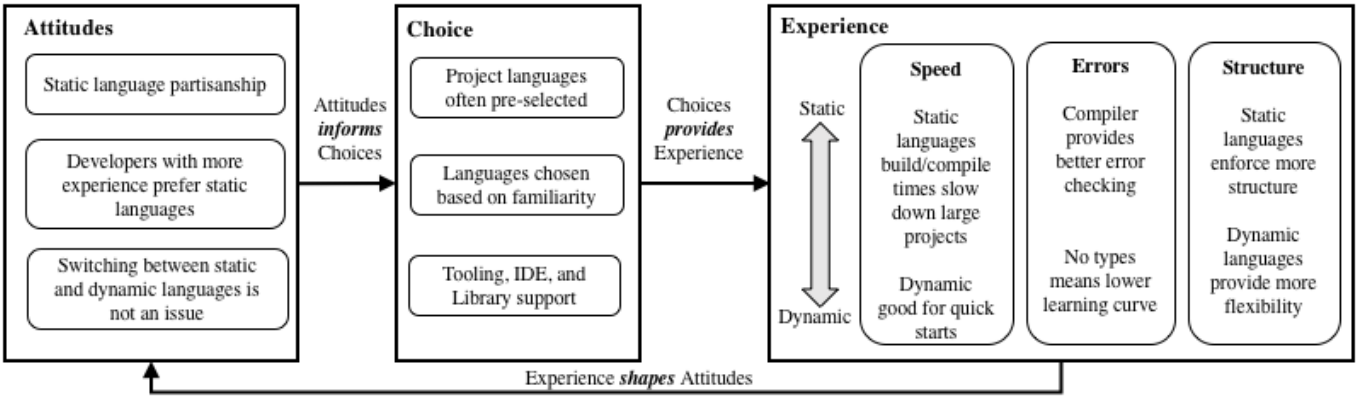

Fig. 1: Theory of static vs dynamic language choice with categories: Choice, Experience, and Attitudes, their relationships, and the factors that influence them. Choice provides experiences, experiences shapes attitudes, and attitudes informs choice.

IV. Theory of Static vs Dynamic Language Choice

The primary theory emergent from the data is that there are several factors that underpin programming language choice, see Fig. 1. These factors can be aggregated into three key categories: *attitudes*, *experience*, and *choice*. In addition to being influenced by their factors, categories can also influence each other, with experience shaping attitudes, attitudes informing choice, and choice providing experience.

A. Attitudes

The Attitudes category represents a developer's existing bias for or against a certain language or class of language, which influences their overall decision making. If a developer has a positive bias towards a static programming language, they are more likely to use it for personal and enterprise projects and likewise if they prefer dynamic languages. This can also hold true for negative perceptions of programming languages. If a developer has a negative perception of a language, this will also impact language choice. There are several factors that can shape a developer's attitudes which include: static language partisanship, switching between static and dynamic languages not being an issue, and that more experienced developers tend to prefer static programming languages.

1) Static language partisanship: Refers to how developers that primarily use static languages feel strongly about the advantages they offer. Several participants indicated that they mostly used typed languages. Due to these languages performing error checking during compile time rather than at runtime, participants felt more secure about their code being error free when executed. These participants also believed that dynamic languages did not offer the same level of error checking, with errors potentially being present in programs using dynamic languages for longer periods of time.

> *"It gives you a better sense of security in the end that you've done something, you can leave it and it's working. If you need to touch it, the compiler will tell you why. There's a sense of security once you run the compiler and it tells you it's ok. With JavaScript, you could have a typo and not notice it for 5 years."* P7

Participants who strongly approved of static languages found that using dynamic languages was not faster and felt using a static language produced greater efficiency and programming speed. Although there was more to type due to having to declare types, participants who strongly supported static languages claimed developers should be fast typists anyway and that the additional time spent was often saved when it came to error checking and fixing bugs. It was commonly stated that dynamic languages were less reliable and that having a type-checking system allowed for better planning and more structured development due to having to consider the type of each object and how it would be utilised.

Those who used dynamic languages were less vocal in their support for static or dynamic languages. They looked at both types of languages equally and considered the merits and drawbacks for both. Participants who primarily used static languages were strongly in support of them and a few stated that they would not use dynamic languages unless there was absolutely no alternative. They frequently cited that many dynamic languages had a static counterpart that would enable them to have the benefits a static language offered (e.g. TypeScript being a static version of JavaScript).

Static language partisanship shows a clear indication of developer's bias towards static languages and against dynamic languages. For many developers, there is an ingrained inclination towards the usage of static languages to the point where dynamic languages are not considered unless absolutely necessary, which is significant in programming language choice.

2) Developers with more experience prefer static languages: Refers to how developers with more experience programming tended to more strongly support the usage of static languages for personal and industry projects rather than dynamic languages. One possible reason for this is that significant usage of dynamic languages has only picked up recently, while participants with more experience will have been programming for companies and projects well before this shift

towards web programming. This would imply that experienced developers have more history with static languages and may feel more comfortable using them.

> *"In my experience, where I have found serious problems is that say I made a typo, dynamic languages don't tell me anything at all. I don't find out until I eventually see that the code is not working and then I check that the spelling is wrong. If I had the ability to pre-declare and if I try to reference a member I didn't declare, it'd immediately throw an exception and tell me to fix it."* P10

Another reason is that more experienced developers are more likely to hold senior roles within project teams, often acting as managers or lead developers. Thus, they may value different traits in programming languages than junior developers or those who focus more on personal projects and start-ups. There may be an emphasis placed on having better error checking or enforcing structure throughout development, both of which static languages provide through having a compiler and type declaration. Some developers may have pre-existing biases which can be built up over a long period of time and tend to be further up within a team or company hierarchy, either as a lead developer or a project manager. This gives them more control over projects and language selection, which may be impacted due to this factor.

3) Switching between static and dynamic languages not an issue: Due to the significant differences between several static and dynamic programming languages, using either for too long may cause developers to forget some of the quirks and idiosyncrasies of other programming languages and make a transition to another project difficult. This was often not the case, with participants indicating that their training both at an educational and corporate level meant that they were well-versed in several programming languages and that alternating between them did not cause an increase in errors or mistakes.

> *"By and large, you've got stuff on the screen to look at and I can switch reasonably well now anyhow"* P9

Some participants stated that although there were minor difficulties such as re-adjusting to the usage of curly brackets and semi-colons if returning to Java from using Python, these were often short-lived once they got back into the "swing of things." Although some developers indicated that they may have a preference for working in a given language due to experience or because they enjoyed how that language worked, this did not impact on their development capabilities in other languages that they used less, but were still familiar with.

> *"You'll be programming in Java & then switch to Python, add a semi-colon, and think that is not right."* P4

B. Choice

The Choice category represents the thought process that a developer undergoes when actually selecting a programming language to use for a project. It is a measure of the factors that influence how much say a developer feels they have when making a language selection and whether or not they have the ability to impact this selection if it is not being made by them. Choice is effectively the final steps in a developer's rationalisation of using a given programming language over another and the impact that they perceive it will have on the development process. There are several factors that can influence choice which include: project languages often being pre-selected, languages being chosen based on familiarity and tooling, library, and IDE support for a selected language.

1) Project languages are often pre-selected: Several participants indicated that the choice of programming language was not their responsibility. All participants were asked whether they selected the language used in the projects they had worked on and for most this was not the case.

> *"It was something that the founder learned and liked. They thought it was good for solving mathematical problems and we've used it since."* P14

Languages were either selected by the lead developer or management or they came onto an existing project that was already using a certain language due to large, pre-existing code bases. This restricted the choices that were available to project teams and meant that there were sometimes few viable languages to choose from. Often, these were languages that the participants already knew. However, in instances where the participant had to learn a new language, management was generally supportive and provided assistance.

Most participants tended to believe that the languages chosen by the organisations and teams they worked within were good fits for the project. Despite not being able to significantly impact the choice of development language, most developers felt that this was not important and that they were usually brought onto projects that fit their skillset anyway. However, there were some exceptions where developers believed that the project could have better met time deadlines, budget, or functional/non-functional requirements if a different programming language was used. Another reason for why programming languages were pre-selected was that the company a participant worked for often specialised in a given language and almost all of their development was done in that language.

For lead developers, project managers and those fortunate enough to be able to have a direct impact on language selection for projects, it was important to analyse how decisions were made. Often, there was less choice available than initially believed due to restrictions such as company expertise or pre-existing code bases that were to be utilised. This was interesting as it showed that both senior developers and those further down the chain of command had this lack of choice in common and was certainly a factor in how programming language decisions were reached.

2) Languages chosen based on familiarity: Many participants indicated that project leads and lead developers often selected programming languages that they were personally familiar with or that they felt the majority of workers within the project team would be familiar with. One reason for this stated by participants is the difficultly in attracting new workers for projects if they were developed using languages that people were unfamiliar with. By opting to use more popular languages (e.g. JavaScript, Java, or C++) it would be easier to recruit

experienced developers for projects. Another reason that fell in a similar vein was that using a language that developers were unfamiliar with would slow down development and make it more costly. This is due to the expense with having to train people in a new language and potential increase in time spent error-checking due to inexperience being more likely to introduce bugs into project code.

> *"There was a lot of ugly code because it was new to me and if I could go back, I'd definitely clean it up."* P3

Conversely, by selecting a familiar language, lead developers felt that development would be faster and result in a higher-quality product. When it came to selecting a programming language for personal projects or projects where they were the lead, developers often opted for languages they were familiar with or languages that were similar. Developers who favoured and were used to programming with static languages were more likely to choose those as was the reverse case.

> *"I don't think any decisions were made about Python because of syntactical reasons. I think they chose Python because everyone knew it."* P12

Languages chosen based on familiarity show how non-technical factors can be a decider in what programming language is used for a project. Often, it is not just the technical benefits and drawbacks that must be considered, but also the benefits to the team. By selecting languages that are familiar to teams, developers believe that they are increasing the odds of success by minimising any particular learning curves.

3) Tooling, IDE and library support: These represent some of the technical factors that may impact why a specific language was chosen. Tooling refers to tool support, which are development tools that can be utilised to support and complement programming languages by providing additional functionality that they do not presently have. IDE support is defined as the set of IDEs that can be used or are compatible with the selected programming language, while library support is the list of libraries and the additional services or functionality that these provide. The support provided for a language can be an influence behind a developer's choice regarding whether or not to use that particular language. Several participants felt that tool support was a major benefit when selecting a language, due to the options it added. Some stated that it simply allowed you to do more than an equivalent language without tool support.

> *"Static languages enables certain tool support that you can't get otherwise or that requires type inference or runtime tracing or what have you."* P9

Having multiple libraries was a benefit that many participants pointed out with similar reasoning to the upsides of tool support. They felt that it provided significantly increased functionality and a wider range of options that could be utilised when programming, with claims of time-saving due to libraries being able to provide code that would otherwise take extended periods of time to figure out and develop. On the other hand, languages with little library support meant that they had little increased functionality and may not be considered.

> *"We chose Java because there's a library for whatever you need to do."* P5

IDE support was less of a factor, with some developers indicating that although they looked for compatibility with mainstream IDEs, they often did not want to use many of the flashier options instead preferred a more simple approach.

C. Experience

The Experience category represents the previous experiences that a developer has had with a given programming language. There are three subcategories: speed, errors, and structure. Each of these has a static factor and a dynamic factor. Speed refers to how language choice has affected speed in previous projects, errors is the error checking experience that developers have had using previous languages, and structure refers to how structured the development process was when using either a static or dynamic language.

1) Speed: Static – build/compile times slow larger projects down. For participants who worked on larger projects, the build and compile times necessitated by using a static language could become cumbersome and have a negative impact on development. This was often due to having a large number of modules being used. Participants found this to be cumbersome and that the time spent waiting for a build to compile could be completely mitigated or removed through the usage of an appropriate dynamic language.

> *"There's thousands of modules now in the project and TypeScript has to compile and it's really slow. We're often running out of memory in some cases, which is a real problem for us."* P12

If static languages can slow down larger projects due to the increased compile and build times, this may impact the decision-making rationale of a developer for another project. Their experience of a static language providing these increased waiting times may make them less likely to employ a static language for a similarly-sized project in the future.

Dynamic – good for smaller projects and quick starts. Several participants raised the idea that dynamic languages were suited for projects that were small in scale or needed a product up and running quickly. Several mentioned both Python and JavaScript as being two languages which were easy to set up and get something working quickly. Often, this was better for personal projects, where participants may not have the time to commit heavily to them and using dynamic languages would result in observable results sooner. Developers believed that this often had an impact on project success as getting the framework up and running quickly meant that work could begin faster and less time was wasted on setting up. This allows for maximum efficiency and allocating more time and resources to the software development stage rather than being bogged down in setup.

> *"Dynamic languages are great for small hacky things."* P9

> *"The setup was super fast. You just have the command line interface, the node package manager and it all just goes. The overall setup did contribute to the project."* P1

2) Errors: Static – provide better error checking. One common trend amongst all participants was that type checking and the presence of a compiler generally meant that they provided better error checking for programs. Errors were caught before runtime and the compiler or IDE would inform the developer if there was an error and what had caused it. This was different from dynamic languages, where error checking does not occur until runtime.

> *"A lot of errors don't show up until they actually happen in JavaScript, C# is a lot clearer since the compiler will tell you if there is an error."* P2

Proponents of static languages who believed it had superior error checking often stated that dynamic languages do not inform you about misspelled items and other basic items, while they claimed that static languages would pick these up immediately and point the developer to the location of the typo due to having a compiler. In addition to this, declaring types meant that any type-associated errors were either eliminated from the beginning as the developer clearly knew what type would go where or the compiler would rapidly pick them up, allowing the developer to fix them.

> *"The times I've dealt with JavaScript, it hasn't been good. It's really not clear what types the inputs are and what the outputs are."* P10

Some participants indicated that using dynamic languages meant that testing and debugging was harder, with this increasing the longer on a project. One reason given was that without types, it was harder to interpret and understand what was going on inside the code. Using a static language provided better clarity and made it easier to look at other people's code when debugging or doing pair programming.

> *"Looking through other people's code to see what's happening is a lot more difficult, especially when one person breaks one thing and find out where the break is being caused. It's even worse there's more people working on it. Using a static language might have reduced this."* P1

Dynamic – easier to learn. Participants claimed that dynamic languages tended to be easier to learn for those new to programming, learning a new programming language, or who had just joined industry. Participants found that not having to declare variables allowed them to get more work done with less effort which minimized the overhead by having to think less about the semantics of their code.

> *"I program a lot faster without types. I find them obstructive to my thought process of continuing to design something. It may be because I design things as I go, rather than planning them out."* P11

Type declaration was another step that typically slowed things down for these developers. When learning a new language, several participants stated that having to understand and declare types would have slowed down the rate at which they learned. This was because they would have to worry about whether variables were correctly typed in addition to learning and applying new skills and concepts. Conversely, with static languages, the concept of typing and how types worked for specific languages meant that there was a greater learning curve and thus, take longer to get working code.

> *"Starting out, it makes it a little simpler. It's a tiny bit of mental labour you don't have to do, meaning you can think at a higher level."* P4

New users of dynamic languages felt that they could immediately make progress on their projects and work without having extra consideration of variable types, while new users of static languages believed that there was more of a gap before they could get something working. A language that participants commonly cited as being easy to learn was Python. Even amongst those who advocated for static languages, Python was still regarded as one of the best programming languages to introduce to those who had never done programming before due to the increased complexity typing brings, and relative straightforwardness of the language itself.

3) Structure: Static – enforce more structure within development. Several participants stated that the usage of static languages enforced structure throughout software development. Due to having to declare the types of variables meant that forethought had to be put into envisaging how the code would look before entering it. Some participants believed that having a more structured development process where they had to put forethought into typing and the overall structure of the program meant that there would be less errors and the overall experience would be more streamlined.

> *"Once we had it up and running and we could show them how everything was organised. In the end, code quality and organisation of code [using Java] was much higher than the JavaScript project we also had running."* P7

Using a static language with a more structured development process impacted the experience of developers. For many participants, this is a matter of personal preference. This shapes a developer's experience as it is one thing to have read about structured development and another to apply it, with some developers responding better to more freedom.

Dynamic – provided more flexibility within development. Some of the participants in the study believed that dynamic languages allowed developers to have more flexibility in the development process, without being constrained by type declaration and other enforced structures that arise from static programming languages. Some participants felt that the ability to ignore typing meant that they could spend more time thinking about how to solve the problems presented by the project rather than having to focus on getting the structure and typing right.

> *"With JavaScript, you can do whatever you want. If you're using Java, you have to adhere to the rules."* P1

Dynamic languages provided more flexibility on code structure and are learned by having previous experience with a language, rather than relying on theory. If a developer uses a dynamic language and finds that it allows them to not have to worry about conforming to rules, and they find this approach works for them, it will build a positive experience.

V. Discussion

We now discuss the relationships between each of the categories, with *choice* providing experience, *experience* shaping attitudes, and *attitudes* informing choice.

A. Choices Provides Experience

The relationship between choice and experience is represented by the choices that a developer makes provides them with experience in the future. With a language choice being made and time being spent using the language for either an industry or personal project, the developer builds a greater familiarity with that language. As this familiarity increases, the developer can examine whether the reasons they used to choose that language were in fact justified and met or if usage of that language did not deliver the results they believed it would. Thus, the choice of language provides experience that can be used for future choices. Each of the three factors present within the choice category have an influence on a developer's experience regarding static and dynamic languages.

Project languages often being pre-selected can bring a negative perception of that language to a developer, if they did not enjoy the development process involving it. They may feel they were forced into using that language and that given their own choice, would much prefer to use something else. On the opposite side, this can also build a positive experience. If a developer had tepid feelings about a language, but had to use it and the learning and management structures were there to assist them and they experienced success, this may change their initial negative feelings and convert them into positive ones. The final case is where a developer is ambivalent to the choice of language and their work did not change this. In this instance, there is no positive experience, but no negative experience and they may objectively look at it regarding benefits and drawbacks in the future.

Programming languages chosen based on familiarity will impact experience depending on whose familiarity it was chosen from. If it was the developer's, then this may reinforce a positive experience as they are using a language they are comfortable with. However, if the language was chosen based on a lead developer or manager's familiarity, this results in a similar situation to the previous factor. There can either be a positive influence, a negative influence or ambivalence, dependent on their preconceptions regarding the language and their experience with the development process.

Tooling, IDE, and library support also provides a developer with experience by helping to make programming easier and to simplify complex tasks. They learn what support a language does have and whether it is relevant to the task they were trying to perform. Languages that possess superior support will provide more successful outcomes and developers will look more favourably upon those languages. If their usage of a language involved tapping into its tooling, IDE and library support and this resulted in a positive result, then the experience provided will be positive. If the support was lacking and inhibited a developer's ability to do work, this will result in a negative experience.

B. Experience Shapes Attitudes

The relationship between experience and attitudes is where a developer's previous experience with a programming language then shapes their preconceptions towards that language. Once a developer has used a language and they have experiences with it, these experiences are looked upon when making future choices. They examine their past feelings and sentiments, which feeds into their preconceptions and personal bias. These subjective personal beliefs form the basis of a developer's attitudes to certain programming languages and types of programming languages. Effectively, if a developer has a positive experience with a programming language, then they will have a positive disposition towards consideration and future usage. However, if the experience was negative, then their perception will impact any future considerations. Rather than looking at empirical research and letting previous studies inform them, developer's preconceptions are instead shaped by their experiences. Factors within the experience category all influence a developer's attitudes regarding language usage.

Speed is an indication of how static or dynamic typing contributes to the overall development speed of the project. The compile/build times of static languages slows down larger projects. This reflects experience as it is often not something a developer can foresee, but something that is noticed over the span of the project as it increases in size and scope. With this experience, a developer's attitudes towards static languages is altered. If compile or build times were increased due to use of a static language, a developer may be less inclined to select a static language for another large project. Contrary dynamic languages are good for quick starts and smaller projects. This represents a developer's experience using dynamic languages for small-scale projects. For projects that require working code or deliverables in a short span of time, developers may be more likely to turn towards dynamic languages if they have previous experience of using them for a similar purpose in the past. This then shapes their attitude towards dynamic languages as they view them as being well-suited to small projects or those which require a quick start.

Errors represents the experiences regarding error checking in languages. The first factor that falls within this subcategory is that static languages provide better error checking. This is an indication of a developer's experience with a static language and whether it picks up on errors. It also covers previous experience regarding dynamic languages and their supposed weakness in error checking. This factor can shape a developer's attitude as it provides a clear comparison between the two types of languages. Previous usage of static languages where errors were identified and caught by the compiler and the developer was able to fix them as a result will provide a positive experience. This would shift their attitude of static languages to a more positive slant. Likewise, if previous usage of dynamic languages resulted in less errors being detected and a longer time spent debugging and cleaning up code, then a developer's attitudes towards dynamic languages will be negatively shaped by their error checking experience.

Structure encompasses a developer's experience of how structured or flexible development is using either a static or dynamic language. Static languages enforce more structure within development which indicates how developer's felt static languages affected pre-planning and overall code structure by enforcing type declaration. Structure can have either a positive or a negative effect on a developer's attitude towards static or dynamic languages, depending on their personal preference. If a developer enjoys having rigid development where everything is planned before hand, it will have a positive impact. Otherwise, it will have either no or negative impact. However, structured development was something that more experienced developer's sought. This was usually because they had more experience and acted as project leads and managers. Dynamic languages, however, provide more flexibility within development. This represented whether developer's believed that using dynamic languages would allow them greater flexibility when it came to structuring their code and if it permitted more coding on the fly. Personal preference was significant when it came to whether or not dynamic languages provided a positive or negative impact. Developers that engaged in lots of personal projects enjoyed the flexibility that dynamic languages brought due to less effort required to consider type declaration and time could be put towards getting results. However, this trait was not valued by experienced developers who had acted as project leads, as having a greater degree of pre-planning usually meant that projects were more successful.

It is clear that a developer's previous experience with a static or dynamic language (be it positive or negative) has a significant influence on their attitude towards that type of language in the future. Effectively, the experience shapes their attitudes and moulds their perceptions and preconceptions of static or dynamic languages. This can either be through validating and reinforcing pre-existing beliefs and biases or by changing them and resulting in adopting new languages.

C. Attitudes Informs Choices

The attitudes that developers have regarding certain languages and types of languages are significant in the choice of language. Sometimes the decision to use a certain language or discount it from selection simply boils down to whether a developer likes that language or not. Attitude is difficult to quantify as it deals with a developer's feelings and there are limited concrete ways of measuring this. If a developer's preconceptions of a language are negative, then they will usually not use that language unless there are significant gains to be made from doing so. Likewise, if a developer has a strongly positive perception of a certain programming language, then they will be more inclined to use that language, even if it is not the most appropriate for a project.

Static language partisanship represents a developer's strong positive bias towards the usage of static languages. Participants who advocated for static languages were strongly in favour of them and strongly opposed the usage of dynamic languages. Whereas those who preferred dynamic languages tended to acknowledge the strengths of dynamic languages but accepted that there were areas where static languages performed better (e.g. error checking). The strong preconception that static language partisanship shows is indicative of how attitudes can inform a developer's choice in what language to use, as those who displayed static language partisanship would be hard-pressed to choose a dynamic language for a project.

Developers with more experience tend to prefer static languages indicates positive bias towards static languages due to their experience. Participants who had less experience in industry tended to prefer to use dynamic languages for a variety of reasons, while participants who had more years of experience tended to opt for static languages. This is another preconception that is held within a more limited group of participants, but can still influence the choice that they make.

Switching between static and dynamic languages was not an issue represents the difficulty a developer may have if two different components of a project are developed using differing languages and their perception of it. For many participants, this was a non-issue as they adjusted rapidly with only a few minor errors being made. However, when making a choice, lead developers may assume that it would be better to have all components of a project use the same language.

A developer's preconceived attitudes towards certain languages or types of languages can impact their choice for a project. If a developer has a negative attitude regarding a programming language, then they are unlikely to select that language even if it is the best suited for a project. The reverse is true for positive perceptions, which may result in choosing a language that is not an optimal fit for a project. Thus, a developer's existing attitudes towards specific languages directly informs the choice of language that they will make.

VI. Conclusion

Our aim was to develop an emergent theory of why developers do what they do focusing on the usage of static or dynamic programming languages by interviewing developers (n=15) and using Grounded Theory [2], [3]. We produced a *theory of static vs dynamic language choice* that discussed three categories that influenced how developers select languages for projects, the relationships between them, and the factors within these categories. These three categories are: attitudes, choices and experience. Attitudes describes the preconceptions and biases that developers may have in regard to static or dynamic languages, while choice is the thought process a developer undergoes when selecting a programming language, and experience reflects the past experiences that a developer has had with a given language. The relationships between these categories was that attitudes informs choice, choice provides experience, and experience shapes attitudes. This forms a clear link between all three categories and how their factors can shape and influence each other. This is a first cut of the theory and there are several potential future avenues such as interviewing more developers, conducting online surveys, and considering other languages aspects (beyond types systems) to study programming language choice which will further help validate our results.

REFERENCES

[1] P. Carbonnelle, "PYPL PopularitY of Programming Language," http://pypl.github.io/PYPL.html, 2017.

[2] B. Glaser, *Theoretical sensitivity: Advances in the methodology of grounded theory*. Sociology Pr, 1978.

[3] J. Holton, "Grounded theory as a general research methodology," *The grounded theory review*, vol. 7, no. 2, pp. 67–93, 2008.

[4] L. Paulson, "Developers shift to dynamic programming languages," *Computer*, vol. 40, no. 2, 2007.

[5] L. Prechelt and W. Tichy, "A controlled experiment to assess the benefits of procedure argument type checking," *IEEE Transactions on Software Engineering*, vol. 24, no. 4, pp. 302–312, 1998.

[6] L. Fischer and S. Hanenberg, "An empirical investigation of the effects of type systems and code completion on API usability using TypeScript and JavaScript in MS Visual Studio," *ACM SIGPLAN Notices*, vol. 51, no. 2, pp. 154–167, 2015.

[7] S. Okon and S. Hanenberg, "Can we enforce a benefit for dynamically typed languages in comparison to statically typed ones? a controlled experiment," in *ICPC*. IEEE, May 2016, pp. 1–10.

[8] S. Hanenberg, S. Kleinschmager, R. Robbes, E. Tanter, and A. Stefik, "An empirical study on the impact of static typing on software maintainability," *Empirical Softw. Eng.*, vol. 19, no. 5, pp. 1335–1382, 2014.

[9] S. Hanenberg, "An experiment about static and dynamic type systems: doubts about the positive impact of static type systems on development time," *ACM SIGPLAN Notices*, vol. 45, no. 10, pp. 22–35, 2010.

[10] A. Pano, D. Graziotin, and P. Abrahamsson, "What leads developers towards the choice of a JavaScript framework?" *arXiv preprint arXiv:1605.04303*, 2016.

[11] L. Meyerovich and A. Rabkin, "Empirical analysis of programming language adoption," *ACM SIGPLAN Notices*, vol. 48, no. 10, pp. 1–18, 2013.

[12] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in GitHub," in *FSE*. ACM, 2014, pp. 155–165.

[13] A. Martin, R. Biddle, and J. Noble, "XP customer practices: A grounded theory," in *Agile*, 2009, pp. 33–40.

[14] ——, "The XP customer team: A grounded theory," in *Agile*, 2009, pp. 57–64.

[15] S. Adolph, W. Hall, and P. Kruchten, "Using grounded theory to study the experience of software development," *Empirical Softw. Eng.*, vol. 16, no. 4, pp. 487–513, 2011.

[16] R. Hoda, J. Noble, and S. Marshall, "Grounded theory for geeks," in *PLOP*. ACM, 2011, p. 24.

[17] ——, "Developing a grounded theory to explain the practices of self-organizing agile teams," *Empirical Software Engineering*, vol. 17, no. 6, pp. 609–639, 2012.

[18] S. Dorairaj, J. Noble, and P. Malik, "Understanding lack of trust in distributed agile teams: A grounded theory study," in *EASE*, 2012, pp. 81–90.

[19] M. Waterman, J. Noble, and G. Allan, "How much up-front?: A grounded theory of agile architecture," in *ICSE*. IEEE, 2015, pp. 347–357.

[20] R. Hoda and J. Noble, "Becoming agile: A grounded theory of agile transitions in practice," in *ICSE*. IEEE, 2017, pp. 141–151.

[21] P. Montgomery and P. Bailey, "Field notes and theoretical memos in grounded theory," *Western Journal of Nursing Research*, vol. 29, no. 1, pp. 65–79, 2007.

[22] A. Pang, "Why do programmers do what they do," 2017, Honours Report. Victoria Unviersity of Wellington, New Zealand.