App-Directed Learning: Support for Learning in Software Development

First Last
Affiliation
City, State, Country
first.last@affiliation.name

First2 Last2
Affiliation2
City2, State2, Country2
first2.last2@affiliation2.name

ABSTRACT

Software developers face a constantly changing set of platforms and technologies that they may need to learn to achieve their career goals. Learning a new platform is a difficult task involving possibly new development tools, programming languages and application programming interfaces (APIs). To explore this learning process, we have conducted an interview and diary study involving ten developers learning a new platform as a side project. We have analyzed these results and characterized a range of learning behaviors which appear to be structured by the application's on-going development. We also explore the implications this may have for development tools in order to explicitly support learning as an integral part of software development.

Keywords

self-directed learning, mobile apps, Windows Phone

Categories and Subject Descriptors

H.1.2 [**User/Machine Systems**]: Software psychology

1. INTRODUCTION

Software developers face the need to constantly learn a changing set of platforms and technologies to achieve their career goals. In just the past decade, the main focus of software development activity has moved from desktop software to web and mobile applications, and on to cloud applications that run on all varieties of devices, including desktops, tablets, and phones. Even developers whose employment does not focus on new platforms may spend some of their extra time experimenting with the new technologies to stay relevant and develop skills for future opportunities. Thus, one of the challenges of being a developer today is that life-long learning skills are required to enjoy a successful career.

Many CS education researchers explore how software developers learn new skills (especially in formal pedagogical settings such as school coursework). End-user software engineering researchers study how non-developers learn to get work-related projects done in non-pedagogical settings. Our research, however, focuses on a different case. How do professional software developers adapt and extend their pre-existing knowledge and skills to learn to build applications on new platforms for their own *non-work-related* side projects? And, with this understanding, how can we enhance existing software development tools with learner-centered features that can help these developers accomplish their life-long learning tasks?

To this end, we conducted an interview and diary study with ten professional software developers who were learning to develop Windows Phone 7 (WP7) apps outside of their regular work (which was not related to mobile software development). An initial analysis of our data [30] identified an existing approach, *self-directed learning* (SDL), that was the key to organizing the primary insights from our data. Using this approach as an analytic lens, we have found that these 10 software developers exhibited a variety of self-directed learning behaviors which resemble those found in problem-based learning (PBL). [29] First, each developer uses the application he or she is building to personalize and structure his or her learning process. What makes this *app-directed* learning approach we observed different than PBL is that the "problem" (i.e. the mobile app) is conceived of and developed *entirely by the learner*, not an instructor. Second, our developers interleaved their learning and application development work on the developers' own schedules. Third, developers revisited topics as needed in order to improve their app. This working process appears to guide the developer about what needs to be learned and provides feedback on whether that material is properly understood.

In presenting the results of our analysis and exploring the implications of those results, this paper makes three contributions:

1. We illustrate how our subjects developed expertise on the WP7 platform and show how they overcame challenges, both pragmatic and meta-cognitive, along the way.
2. We introduce an organizing principle called *app-directed* learning to describe how our study participants structured their self-directed learning process.
3. We illustrate designs for new learner-centered [34] software development tools that apply models of the platform domain and user profiency with platform concepts to help developers overcome their learning challenges and more effectively gain mastery over new platforms and ship their apps.

In this paper, we first provide background on self-directed learning, learner-centered approaches to pedagogy, including problem-based learning, and highlight the differences between professional and end-user software developers in the ways that they learn new skills. We then describe our study methodology and the findings from the analysis of our study participants' interviews and diaries. We then compile our observations together, and introduce the main tenets of app-directed learning. Finally, we explore how domain-oriented models for platform development and developer expertise can be used to improve the design of software development by explicitly supporting app-directed learning activities.

2. BACKGROUND

In this section, we introduce approaches to understanding learning that have influenced our study, analyses, and proposals for new tools. We focus primarily on self-directed learning, as we use this as a key analytic lens to frame our work. We then show how these

principles and those of learner-centered pedagogy are combined in a more concrete form called problem-based learning. Finally, we see how end-user software engineering research extends problem-based learning to self-directed learners, who define their own tasks and get them done without the benefit of an instructor scaffolding the learning process.

2.1 Self-Directed Learning

Adult, self-directed learning (SDL) was first described by Tough and Knowles, both students of Houle who established the field in the 1960s. Building on Houle's proposition that adults are motivated to learn because of personal enjoyment [17], Tough discovered that learners like to be self-directed and plan their learning projects themselves [37]. Knowles organized these and other concepts into a theory of adult learning called andragogy [23] which has six assumptions. 1) Adults need to have a reason to learn, 2) Experience and error is the basis for learning, 3) Adults take responsibility for self-planning and self-evaluation of their learning, 4) Adults are mainly interested in learning personally meaningful subjects, 5) Adults focus on problem-centered learning more than content-oriented learning, and 6) Adults learn more because of intrinsic motivation than extrinsic motivation. After Brockett and Hiemstra compiled a summary of subsequent empirical data and extensions to the original theory [7], the Internet arrived to shake things up. Draves said that the Internet enhanced learning dramatically because people could learn at the peak times of their own days, at their own pace, and focus on just the areas they wish to learn or that they need the most help in learning [11]. Hiemstra also found a lot of positive benefits with online learning. People used the Internet for self-discovery as well as self-improvement, and learned a lot more a lot more quickly online than they had before [15]. However, people also suffered from information overload, and found it was harder to adapt a wider variety of resources to help them learn in their own tasks.

With the advent of Massively Open Online Courses (MOOCs), SDL in Computer Science has become more popular with the "inverted classroom" [13]. Instructors who have experimented with learner-driven educational styles have had varying degrees of success, often due to a lack of student motivation [36, 18]. Adult learners appreciate SDL more [41], which echoes what we learned from our participants. In fact, Zander *et al*'s study of professional software engineers is the closest related work to the study portion of our project. Their work is oriented towards improving undergraduate education, whereas ours focuses on helping those computing professional become better self-directed learners.

2.2 Learner-Centered Approaches

Self-directed learning aligns well with constructivist principles of learning first postulated by Piaget [33] (*i.e.*, knowledge is constructed by the learner, not transmitted from the outside), and extended by Vygotsky [39] (*i.e.*, learning is easier with the help of others) and Papert [31] (*i.e.*, learning works well when building artifacts). These ideas, which also appear in the tenets of SDL, have been explored in many programming environments designed for children and novice programmers, such as Logo [31], Alice [12], Scratch [25], and others [20]. Related environments have supported instructor-provided scaffolded interactions that provide structure to self-directed students learning on their own initiative [3, 19].

The pedagogical design principles that these environments have in common, termed *learner-centered design* [14, 34], are well-suited for implementing problem-based learning (PBL) pedagogies [29]. Using PBL, students can focus on completing an explicit learning goal that combines a complex set of learned skills. We are inspired by learner-centered design and PBL in the mechanisms we propose to augment software development tools to support app-directed learning for adult, professional developers. By working with pre-existing, professional tools, our ideas can have greater impact on the existing population of adult, self-directed learners in software engineering (and perhaps, all professional software engineers).

2.3 Learning Software Development

As we have stated previously, our study focuses on individual, self-directed learning by software development professionals who accomplish self-defined non-learning-related goals by building their own applications outside of their regular work. Brandt *et al.* studied how professional web developers use web-based resources to learn new ideas, even before they fully understand them, discover more detailed information to refine existing knowledge when at first they did not deem it necessary to understand it, and remember things they saw, but were not valuable enough to remember [6]. Dorn and Guzdial studied a similar kind of population, *end-user* web developers, who are untrained in software engineering and do not consider themselves software engineers, yet may have extensive domain expertise in web development. Building on Brandt *et al.*'s work, Dorn and Guzdial found that end-user web developers differ from professional web developers in their reduced knowledge of computer science concepts and processes [10]. However, both groups utilize similar knowledge acquisition and learning styles: both learn on-demand, only when necessary to accomplish a task; both opportunistically read whatever online and offline sources are discoverable and available; both spend a lot of time determining what information they read is relevant, necessary, and sufficient to complete their task; and, both find it difficult to assimilate advanced domain concepts before they have the prerequisite knowledge to understand them.

The mobile application software developers we studied used similar learning styles to these web developers, but their extensive, prior knowledge of computer science and alternate platform concepts gives them enhanced abilities to make complex applications and gain deeper understanding through revisitation of previously learned platform concepts.

2.4 Mobile Application Development

Mobile phone platform providers such as Google, Apple, and Microsoft supply app developers with Software Development Kits (SDKs), which include platform APIs, development tools, associated documentation, and online Q&A forums. For example, Windows Phone (WP) app developers make use of the Visual Studio IDE to access C#, C++, and Visual Basic platform APIs and class libraries. These tools enable them to design XAML–and HTML5–based user interfaces, to test and debug their apps using a desktop computer-based phone simulator (giving them access to a variety of phone form factors), and provide them with a set of tools for deploying apps to the Windows Phone App Marketplace. Developers also have access to thousands of pages of Microsoft-provided documentation including API reference documentation, overviews, tutorials, *how to* documents, and online forums in which they can learn about the platform and its technologies.

Developers also use resources created by members of the broader development community unaffiliated with the platform owner. Examples include blogs, online forums, and question and answer sites such as Stack Overflow. Recent research has explored the ways these resources are used to answer developers' questions [6, 38], their effectiveness at covering features of a platform [32], and their value as a source of examples for developers [5, 28, 40].

3. STUDY

We conducted a study with 10 professional software developers who were building Windows Phone 7 applications in their spare time away from work. In addition to learning the platform in order to prepare for future career opportunities, the participants expressed a desire to "work on something end-to-end" (P3) rather than working on his more narrowly defined full-time software development job, and to build apps that people would find useful rather than learn the platform by developing "test apps that no one is going to use." (P7)

Our participants' development and learning sessions were intermittent and often brief (around 2 hours per session). Our participants said that they viewed learning the WP7 platform as a long-term career investment. The participants were fairly experienced, with a mean of 9.4 years of professional software engineering experience (SD: 6.6 years), but all had much less experience with the Windows Phone 7 platform (Mean: 1.2 years, SD: 9 months). All but one subject had previous experience with Visual Studio and C# programming. Two had programmed in Java before, one for the Android mobile platform. Two had experience with Visual Basic, two knew SQL Server, and three had programmed for the XBox in C++. Only one knew the XAML interface language before starting WP7 development.

3.1 Methodology

Upon entering the study, each participant was interviewed by one of the authors for an hour. These semi-structured interviews focused on our participants' experiences learning to develop on the platform. During the interviews, we also asked our subjects to describe and characterize their level of expertise with the WP7 platform and explain how they achieved it. Our questions focused on specific learning and development episodes and explored details around what was learned, as well as how. We also discussed particular learning challenges our participants faced.

The participants were also asked to keep a diary of their platform learning experiences twice a week for two weeks. Each diary entry answered the same question: "What is one thing you have learned recently about the WP7 platform, and how did you learn it?" The goal of the diary was to uncover the day-to-day learning experiences and use of learning resources while those experiences were still fresh in the our participants' minds. In total, we received 2 to 4 diary entries per participant, for a total of 24 entries.

Four participants were interviewed again two weeks later. Unfortunately, the recordings for two of these participants were lost prior to being transcribed. All of the first round interviews were listened to and transcribed by both authors. The second round interviews were transcribed by a paid service.

3.2 Analysis

We used a Grounded Theory approach [9] to analyze our data, beginning with open coding. Our initial analysis focused on identifying concepts related to what was learned, how it was learned, and relevant challenges. We spent some time refining our codes until we agreed on their definitions. Then we moved into a theoretical coding phase to try to explain what we saw in the data, the codes, and their interconnections.

Based on our initial analysis, we found that the learning approaches of our participants resembled aspects of self-directed learning [24]. As a result, we have used it to formulate analytic questions to refine our analysis and organize the discussion of our results.

1. How do our participants determine what to learn?
2. How are immediate learning goals formulated?
3. What resources are used, and how are they found?
4. What learning strategies are used?
5. How are learning outcomes evaluated?

In addition to the above five questions, we also found one additional question to be useful in our analysis: *What motivates our participants to continue to learn?* Answers to this question capture some of our participants' initial and continuous reflection on whether learning and developing their apps is worthwhile, despite the challenges they face.

4. FINDINGS

The combination of learning behaviors we have observed can be characterized as an instance of self-directed learning, which we call *app-directed learning*. Our discussion of app-directed learning is organized around the six analytic questions mentioned above.

4.1 Selection of Learning Topics

Newcomers to a platform need a way to determine where to start in their learning, and even to learn what "capabilities the platform provides" (P7). Deciding on an application to build helped our participants move forward. Once they decided on an application, they could figure out which parts of the platform they needed to learn. As participant P10 designed his application he determined "all the key mechanisms ... and implementation artifacts" he needed and used that design to structure his learning and figure out "what I don't know". One can consider the application and the features required to build it as a *self-defined, personalized syllabus* that structures each participant's platform learning process.

To illustrate this point further, consider Participant P3's streaming music application. His personal syllabus included learning the WP7 APIs related to web services, streaming data, and playing foreground and background audio. None of those audio APIs were relevant to any of our other participants. A consequence of this specific learning path is P3 expects his platform expertise to be incomplete. "Becoming a platform expert is not my goal, but probably will be the result — at least for subsets of the platform."

Not only do participants focus on different topics to learn, but the need to build a working application helps them determine the best order in which to learn them and even after similar amounts of learning efforts, our participants depth and breadth of knowledge varied significantly. Participant P4 learned an early set of skills during a hack-a-thon, but found that "there are still lots of things I could use help with ... stuff I didn't pay too much attention to." Participant P9 "can develop basic applications" but has "some issues in [his] understanding of how you structure the project and how you develop an application that is highly responsive" because he was developing "features on the fly." Participant P10 organized his learning based on "the sequence of problems that I encounter" staring with what happens when running the app "what's the first problem that I'm encountering?"

The key finding here is that rather than talking about how someone might learn the WP7 *platform*, it may be more relevant to consider how someone might learn to implement *a particular set of application features* on the WP7 platform. A developer who wishes to learn to implement "some new cool features of Windows Phone" begins by asking "what the first step would be" and learning just enough to implement a "simple prototype" (P3, P7).

One significant challenge caused by each participant having a unique syllabus is that there is no one place that he can go to find everything he needs to learn. Even if he could find bits and pieces here and there, the information would certainly not be structured

in the way that would help him to most effectively learn it. From the platform owner's perspective, developing universally applicable platform learning materials and tutorials is almost impossible because every developer's learning needs are, in fact, so different.

Key observations:

- The app that the developer chooses to develop is a self-constructed, personalized syllabus that structures what he will need to learn and the order in which he picks up platform knowledge.
- The app determines and limits how deeply and how broadly a developer will dive into the details provided by the platform. This results in uneven developer knowledge.

4.2 Formulation of Learning Goals

The development process recounted by our participants takes the form of interleaved coding and learning activities in each programming session. Their primary goal is to work on or complete a feature for their application. As they program, they inevitably run into obstacles that block their progress. When they "get blocked" (P6), developers must switch into "learning mode," in which they need to figure out "why is this not working for me? How am I going to accomplish what I need to accomplish?" (P10) and learn it well enough "to get something to work" (P2). These new learning goals require narrowly focused and challenging information-seeking activities to learn the right information and make progress.

For example, while developing his app participant P10 found that "it's very jarring for the user to transition from the intent to launch your app to seeing your app start" and decided that making this transition smooth was the "first thing" he was going to learn how to fix during his development session. At times he found that much more time was required to learn what he needed to learn than to do the development itself.

> "'Oh, here's the task I'm going to do today,' and then that takes three days or just three times as long as however I thought just because I get bogged down into trying to figure out how to get this one thing working that caught me off guard, or I end up digging through a lot to try to find the answer to a problem." (P10)

Similarly, participant P3 found that learning was time consuming but valuable, even if he did not make any progress on his application. During some of his development sessions he didn't "actually write much code" but instead "I mainly took existing code and moved it around, simplified it, tried things I had read about, and examined it in the debugger to confirm my understanding."

In a sense, our participants' most pressing learning goals are not *learning about API features X and Y*, but in fact, are the much more difficult goals, *learning how to use API features X and Y* ***in my application***. Due to the degree of context-specificity required, these kinds of on-the-spot learning goals are more difficult for a developer to accomplish. In a self-directed learning context, providing the developer with the right help at the right time without having any idea what "curriculum" the developer is following (if any at all), is a fundamental challenge.

Key observations:

- Development is interleaved with learning and is triggered just-in-time when blocked on a task. Developers try to learn the minimal amount required to get themselves unblocked.
- In a programming session, a developer's primary goal is to complete a feature of his app. Learning how to implement the feature is secondary, but is required to make progress.

4.3 Learning Resources

Learning activities often began when our participants got blocked during development. In these cases their process was primarily about searching the web and commonly encountered resources authored by members of the development community rather than official resources provided by the platform owner; "I just search online, and usually the first few results are Stack Overflow" (P9). The goal for some of this searching activity is to find "someone else who has encountered this problem" (P10). Participant P1 "may spend 2 hours just doing Bing searches" in order to find out "how do other people do it." Our participants found that much of the official documentation was limited because it was missing the accumulated experiences of the developer community which provide insights around "ways to use [platform features] you may not have thought of, when not to use it, a-ha moments."

While many good resources are available, at times our participants struggled to find what they needed for their particular context. Finding the right resources might require days of online research. When Participant P3 was dealing with a "rare bug" that few others had encountered, he was unable to find online information that helped. It took him months to solve the problem. Some participants were not able to find answers initially because they did not yet "know the right terms," and searches on naïve terms like "'how do I get that animation when I touch...' turned up nothing" (P3).

Despite these challenges, in many cases it was easier to resolve issues online than to debug the issue using the source code and development tools, due to their limited understanding of the platform. Such a process often begins with "searching for the error message" to "dig up what was causing the problem" (P4). For example, when Participant P1's application threw an exception that he could not explain, he spent two days trying to figure out the reason why. He tried to code workarounds, looked for clues online, and looking at the API documentation to understand "why he is hitting [the exception]" a problem he was sure was "something small and stupid". Many of the explanations he found online were too generic to give him any guidance in his particular circumstance. He eventually found a Stack Overflow post online that suggested a fix involving changing the visibility modifier of one of his classes. A one-line code change solved the two-day-long problem.

Code examples found online or distributed with the SDK play an important learning role. Participants usually started an application by "following the vanilla examples" (P3), and then adapting them to work in the desired application context. Similarly, in a diary entry, participant P4 described his development session as "minimal time spent actually writing code," as he was "primarily altering the samples to fit [his] needs."

Sometimes, however, examples taken from the web just did not work when participants put them in their applications. They found it was time-consuming to find and try out multiple examples, just to discover the "one that works for me" (P10). When Participant P9 tried to use a code example, he ended up "confused between the documentation" and the application's actual incorrect behavior when he tested it. Once he realized that "the code had a bug," he had to look around online for other examples until he found one that worked.

Though time consuming, testing, incorporating, and adapting external code examples, or "toying around" (P4) with examples is an essential learning activity. Participant P6 found that "customizing [the example] to your need would help you understand it a little more" and that when you "try a little different things, that's when you actually see how things work."

Key observations:

- Developers new to a platform, turn to experience-based information written by other developers instead of debugging problems in the source code using debugging tools.
- Though there is no single place for an app-directed curriculum to be found, resources are available to help developers get past many of the small problems that block their progress in a development session.

4.4 Learning Strategies

When first learning about a topic, our participants tried to learn as little as they could get away with, say just enough to get a feature working, and would only learn more when revisiting the same topic later to address an issue that left them dissatisfied with their original attempt. Participant P4 tends to "do a bit of a rush through when doing a first pass of implementing new functionality," even if it is not completely integrated with the rest of the application. Generally, the approach we observed involved "go[ing] through it gradually" (P9), accomplishing one small task after another.

Just getting something to work in an application is not enough to understand the technology completely, or even to know the *best* way to do it on that platform. Participant P2 found that after completing a feature he did not always "know if he had done it right" just that he thought it "kind of works". Similarly, participant P4 felt that he is "not an amazing developer" because even though he can usually "make things do what [he] wants" he depends on lots of "copy and pasting" of code that he does not always understand which means that he can not "create them from scratch."

Participants were often triggered to improve the implementation of a feature in their application and to increase their level of understanding when they became dissatisfied with some aspect of their application, such as slow performance or bugs, or ran across best practices they had not known about. Participant P3 became an expert on one aspect of the API by unknowingly "shipping with odd problems." Over time, he solved those problems "by figuring out how to handle all of the exceptions that [could] arise." However, even after shipping a more stable version of his app, he still felt that he had not grasped certain general platform topics. Similarly, after shipping a first version of his app participant P4 spent time learning best practices that he "didn't go through in [his] first iteration" so that he could simplify his app by making better use of the platform.

Our participants often had breaks between their development sessions because building their app was a side project rather then their main job, which naturally lead to forgetting what they had learned and how they had implemented a feature. There are also times when something is learned, but forgotten because it is not used frequently during development:

> "I may learn it good enough, but then I forget it because I don't use it any more" (P2).

Similarly, Participant P10 said that if he learns something that he can not put into his application immediately, he will need to go "back and relearn that again." As a result, he tries to learn only as much as he can apply at that moment, iterating further as needed when he next returns to learning and coding.

Key observations:

- Developers are triggered to revisit a topic to learn it more deeply when they become dissatisfied with their current implementation, often because a problem with the app is revealed that they want to be able to fix, or because their increased knowledge of the platform inspires them to improve the app.
- Gaps in development sessions due to paying only intermittent attention to their application increase the chance that developers will forget what they learned in the last session, and will have to revisit the topic to learn it again.

4.5 Evaluation of Learning Outcomes

We asked our participants to characterize their level of expertise in order to learn how they self-assess expertise. We found that while their notion of expertise varies, it is consistently and closely tied to the relationship between the amount of time and learning required to actually build something with the platform. The most common attribute they associated with people rated 10/10 in expertise was that they would be able to code a full feature without having to look things up. An expert "can put something together really quickly, [and] use more options and tools" and does not "need to look at docs" (P5). Though our participants might "know where to go to find that out," if they were true experts they "should have had an at-hand knowledge versus having to go and check a forum" (P4).

Our participants often spoke about their personal expertise on particular aspects of a platform rather than more generally. Participant P9 wanted to gain enough expertise to make his app's performance scale well with the amount of data it processed. Participant P3 learned about the background audio API over time by "figuring out how to handle all of the exceptions that can arise" and he felt that this is one of the few areas of the platform that he has deep expertise in.

Our participants evaluated the learning outcomes of specific development episodes based on the progress they were able to make in developing their app; "I have four screens to build, and I've built two out of four" (P10). Learning was often triggered by a development challenge and so in this context learning is successful when the challenge can be overcome. Interleaving learning and coding in this way provides both practice with what is learned and feedback on whether it was truly understood:

> "At the beginning I read through it but it didn't sink in the first time until I actually used it. Actually debugging through it, adding code to it." (P6)

Of course, as mentioned earlier this feedback does not necessarily indicate that the topics are deeply understood, but simply that they are understood well enough for the developer's current goal. After that they tend spend "time trying to move to the next phase and try to finish up the development for the other stuff" (P6), preferring to make continued progress rather than immediately learn more about those topics.

When participants do revisit topics and learn more about them it is because of application issues or because their goals for the app are changing. In these cases they realize that they need a deeper understanding of some topics to fix issues with the application or to make "it a great application" (P4).

Key observation:

- Learning requires practice and feedback. Developing and shipping an app provides a place for both to occur.

4.6 Learner Motivation

Our results reflect on two aspects to this analytical question. The first is about what motivates our participants to continue through learning and development challenges rather than to quit. Not being able to make progress and "solve my problems quickly" at any learning stage can be "a big morale deflator" and spending a significant amount of time learning rather than "in-the-zone productivity

efforts" can lead to frustration and (at least temporarily) quitting or "putting the thing down" (P10). Participant P7 found that spending lots of time learning can lead to the "spark fading."

In particular, it seems that *initial* progress (as measured by getting aspects of the app working) is important for maintaining motivation. Participant P2 gave up on a learning and developing with a new platform (iOS) because he found that "everything was just so much different" and that he had "no idea" how to get started. Similarly, participant P1 had trouble getting started with a new platform and "threw it away" deciding to "try it again in a few years."

The desire to make continuing progress on the app partially explains many of our findings above. When the user is blocked, he learns the minimal amount of material that helps him make progress. This style of learning and coding sets up a feedback loop where the developer is directed towards what to learn (or is made aware of what does not need to be learned), and in seeing its realization in his app, understands how well he learned the material.

A second aspect of this analytical question is: *what drives our participants to continue once they have shipped a version of their app, and their initial development goals are reached?* As we discussed earlier, our participants' app-directed learning defines a set of topics that need to be learned. Some of these topics are revisited as problems arise with the implementation, presenting opportunities for deeper learning. However, we also found that the developers' ambitions for their apps often grow as they learn more about the platform and enjoy some initial success in developing an app. For example, participant P4 after shipping a first version of his app worked to learn more about the platform and what makes it unique to find ways to "make the app shine" which would require "becoming great at a platform and knowing the intricacies."

Key observation:

- The desire to maintain progress motivates a just-in-time learning approach.

4.7 Summary of App-Directed Learning

We found our participants' learning to be interrupted, occasional, just-in-time, interleaved with development, iterative, and highly contextualized by the app being developed. The ten key observations that we pointed out above help explain the *app-directed* approach our participants employed in their self-directed learning process.

Through our *app-directed* viewpoint, we see that our participants simultaneously develop an app on a platform and learn how to develop on that platform. The app they develop plays a central role in structuring their learning activities; it serves as a syllabus for their learning by determining which platform topics they need to learn and also influencing the order in which they learn those topics. To find resources to support their unique learning plan, app-directed learners must explore, interpret, and assess a wide variety of resources, the best of which capture the overlapping experience of other developers.

One might ask why our participants did not avail themselves of other useful, pedagogically-oriented learning approaches, such as building their app using an educational programming environment, such as App Inventor for Android [27] or TouchDevelop for Windows Phone [35], or by taking an online course. But, these environments are designed mainly for novice developers to learn mobile software platforms, while our participants' primary goal is to build a specific, self-defined mobile application. Our participants do not identify as students, do not work with an instructor, and do not desire a well-rounded education in the platform. For our participants, learning begins when they encounter a programming obstacle, indicating the need to learn more about some aspect of the platform to make progress. This just-in-time learning is highly contextualized to the app being developed and can lead to challenging information-seeking activities to find relevant resources and adapt those to the specific problem at hand (also reported by Brandt *et al.* [6]).

App-directed learners manage their learning activities so as to maintain development progress on their app (and maintain their motivation to learn) as they face obstacles. Often maintaining progress means that they learn just enough about a topic to get something to work, saving deeper learning for later, when improvements to the app require it. A consequence of this approach is that learning is uneven; learners vary significantly in what they have learned and how deeply they have learned it.

App-directed learning provides a context for both practice and feedback. Attempting to learn only what is needed to make progress allows learners to immediately practice what they have learned by applying it to the problem at hand. Running, testing, and debugging the application gives the learner immediate and realistic feedback on how well he has understood the topics.

In the next section, we present ideas on how we can better support app-directed learning in software development tools. First, we introduce concepts that we can use to model, with the platform provider's help, a developer's growing proficiency with domain knowledge. Then we discuss how tools can use these models can be used in the redesign of development tools. This discussion is organized around several programming scenarios relevant to our findings.

5. LEARNER-CENTERED TOOLS

There was such a tight coupling between our participants' development work and learning, that we noticed deficiences in their software development tools that hindered their learning progress. We have brainstormed ideas to transform software development tools to better support app-directed learning. These learner-centered software tools are designed to support self-directed learning in the context of developing software.

Fundamental to the tool support we are exploring are two connected models: a model of platform-related concepts (built on top of pre-existing models of programming) and a proficiency model that can track the developer's acquisition and evolving comprehension of platform knowledge. Developing a platform model would require significant effort by platform owners, but the burden could be eased by building on existing pedagogical models of programming [8] and software engineering [4] used for intelligent tutoring systems, and through automated mining of platform-specific information of the kind that have been used for reverse engineering [2]. The profiency model needs to be based on a history of the concepts or modules with which the user has some type of experience. Tool-observable experiences with a topic may include: reading about it, using it in code, debugging code that uses it or shipping an app with such code. Similarly, facing, and ultimately dealing with certain exceptions and the ways in which a user is using the platform can be used to inform the expertise model. The temporal dimension needs to be considered as well, possibly by way of a function that detects practice, to increase the developer's predicted expertise, and a decay function that captures the decrease in developer expertise that comes over time without practice.

Pre-existing software development tools could be adapted to support app-directed learning by enhancing them to gather data to put into the model, and by exploiting the model to provide more explicit help for the streamlining the developer's learning process, and offer guidance for more efficient development. We discuss these ideas in the next section.

5.1 Tool Support for App-directed Learning

The following are five scenarios drawn from our study that show how software development tools can help their developers based on concepts from app-directed learning.

- *Maintaining progress.* When a developer is struggling to make progress on his app's feature, he can get discouraged. A tool could estimate how much progress is being made by tracking the learning objectives it infers are required to build the app. When developers have made little progress for some period of time, the system could help them find the next "easiest" topic to learn and use, by reasoning about the difficulty level and dependencies between the platform topics that make up the app's learning objectives. In this way the tool could it could suggest to the user the next lowest-hanging fruit in order to get him moving again.
- *Getting ready to ship.* We found that when an application is shipped (i.e., published in the platform's app marketplace) and purchased by users, it is exposed to many different operating contexts than developers were able to test for. This often reveals issues with the app that are caused by the developer's limited understanding of various platform topics. As part of preparing to ship an app (or helping the developer decide whether it is ready to ship), a tool could present information about how well relevant platform topics were understood by the developer at the time various pieces of the app were built. For example, code that was written at a time when the app developer was still learning to apply major platform concepts may need to be revisited to squash latent bugs before it should be shipped.
- *Resuming a task.* Our participants worked on their apps on-the-side, and sometimes went days, or even a week or two between development sessions. After such a long break, resuming a task can be difficult from both a development and a learning perspective. Mylyn, which allows a user to save and restore task context, is designed to help with this resumption challenge [21]. We can extend this concept by enabling the tool to monitor learning activities as well as progress towards app completion. This would enable the tool to help the user restore not only his working context, but also his learning context, making it easier to review relevant material as needed and more easily pick up learning from where he left off.
- *Learning and debugging.* Exceptions or other errors were often the triggers for our participants to switch to learning mode. Analyzing and reasoning about the errors is often challenging, especially when the developer is still working to more deeply understand and apply topics he learned earlier. A concept location tool [26] that can map between errors, exceptions, and call stacks and platform concepts, along with a model which knows which app objectives have been reached by the developer, may be able to infer the developer's particular knowledge gaps. It can then suggest learning activities that may help the developer learn how to deal with the error.
- *Searching and browsing platform documentation.* In addition to basic reference material, documentation provided by the platform owner includes thousands of documents, such as tutorials and overview guides. Some of these are appropriate for developers that are working on *understanding the meaning of the topic* while others are more suited for developers who have learned to *analyze and reason about the topic*. By following the dependencies between the platform concepts in these documents, a tool with platform knowledge and the user proficiency model could make it easier to find appropriate material. One way to manifest this in the user interface is to highlight documents on a topic that are at the right level for the developer, and suggest other documents that could be read before and after to augment his understanding of the concept.

In addition to these specific examples, there is benefit in a tool that makes the model's inferences about user profiency and platform concepts explicit in the interface. One popular example for demonstrating the learning path of a person learning elementary math skills is used by the Khan Academy [22]. In our domain, while a developer performs a development task, he could be presented with a dependency map of what has been learned and what still needs to be learned for his task. It could then highlight the leading edge of the map to show what information is best to learn next. A developer presented with such meta-cognitive information may be able to make better use of his learning time. Such information may also increase a developer's self-efficacy and motivation even when his development progress is less than he had hoped.

In addition to augmenting software development tools, we also consider changing some existing features to prevent them from thwarting potential learning opportunities. For example, newer development tools aim to fully, or partially, automate tasks for developers. Our results suggest that there can be downsides to this approach, as it may eliminate some learning opportunities. For example, the learning outcome *applying a concept in a novel situation* from Bloom's Revised Taxonomy [1] can be achieved when a user needs to (manually) adapt sample code. Thus, automating the adapting of examples may prevent this learning [16]. A model of proficiency may provide tool designers the ability to vary the amount or kind of support provided to better balance task progress support with the accomplishment of learning objectives.

6. CONCLUSION

We have conducted an interview and diary study exploring how developers learn a new platform. The study focused on developers learning a new platform as part of a side project rather than their day job. Our analysis has led us to describe an *app-directed* set of self-directed learning behaviors and to explore how they can help inform the design of tools for supporting learning activities.

The key observations from our study are that the features of an app being developed defines the topics the learner needs to learn and influences the order of learning. Often, leaners aim to only learn as much as was needed to complete their immediate development goal, returning to the topic later to gain a deeper understanding (if and when needed). This approaches supports the developer in making continued progress on his app development goals, managing the amount of learning needed for those goals.

Interleaving learning and coding activities provides an opportunity to practice and apply the topics being learned, and to get feedback on how well those topics are understood. It also means that learning activities are highly contextualized to the needs of the app being developed, enabling learners to grow their expertise only in particular topics, and at varying levels of depth, rather than with the platform in general.

Our analysis has implications for the design of software development tool support for learners. Given that coding and learning are so tightly coupled, we have explored how a learner-centered approach to the design of development tools can help with both learning and development. The tool support we explore is based on a model of platform concepts and user proficiency. We argue that this kind of augmented tooling can be useful in a range of development contexts.

7. REFERENCES

[1] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, and M. C. Wittrock, editors. *A taxonomy for learning and teaching and assessing: A revision of Bloom's taxonomy of educational objectives*. Addison Wesley Longman, 2001.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley Longman, Boston, MA, 2003.

[3] C. Bereiter and M. Scardamalia. *The Psychology of Written Composition*. Erlbaum, Hillsdale, NJ, 1987.

[4] P. Bourque and R. E. Fairley, editors. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE Computer Society, 2014.

[5] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer. Example-centric programming: Integrating web search into the development environment. In *Proceedings of CHI*. ACM, 2010.

[6] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of CHI*. ACM, 2009.

[7] R. G. Brockett and R. Hiemstra. *Self-Direction in Adult Learning: Perspectives on Theory, Research, and Practice*. Routledge, 1991.

[8] R. Brooks. Categories of programming knowledge and their application. *Int. Journal of Man-Machine Studies*, 33(3):241–246, Aug. 1990.

[9] K. Charmaz. *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*. SAGE Publications, 2006.

[10] B. Dorn and M. Guzdial. Learning on the job: Characterizing the programming knowledge and learning strategies of web designers. In *Proceedings of CHI*. ACM, 2010.

[11] W. A. Draves. How the internet will change how we learn. **`http://www.williamdraves.com/works/internet_change_report.htm`**, 2002.

[12] W. Dunn, S. Cooper, and R. Pausch. *Learning to Program with Alice*. Prentice-Hall, 2006.

[13] G. C. Gannod, J. E. Burge, and M. T. Helmick. Using the inverted classroom to teach software engineering. In *Proceedings of ICSE*, pages 777–786. ACM, 2008.

[14] M. Guzdial, Y. B. Kafai, J. M. Carroll, G. Fischer, R. Schank, and E. Soloway. Learner-centered system design: Hci perspective for the future. In *Proceedings of DIS*, pages 143–147. ACM, 1995.

[15] R. Hiemstra. Is the internet changing self-directed learning: Rural users provide some answers. *International Journal of Self-Directed Learning*, 3(2):45–60, 2007.

[16] R. Holmes and R. J. Walker. Systematizing pragmatic software reuse. *ACM Trans. Software Engineering Methodology*, 21(4):1–44, Feb. 2013.

[17] C. O. Houle. *The Inquiring Mind*. University of Wisconsin Press, 1961.

[18] V. Isomöttönen, V. Tirronen, and M. Cochez. Issues with a course that emphasizes self-direction. In *Proceedings of ITiCSE*, pages 111–116. ACM, 2013.

[19] S. L. Jackson, J. Krajcik, and E. Soloway. The design of guided learner-adaptable scaffolding in interactive learning environments. In *Proceedings of CHI*, pages 187–194, 1998.

[20] C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Survey*, 37(2):83–137, June 2005.

[21] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of SIGSOFT/FSE*. ACM, 2006.

[22] S. Khan. The Khan Academy. **`http://www.khanacademy.org`**, 2014.

[23] M. Knowles. *Self-directed Learning*. Follett Publishing Company, 1975.

[24] M. S. Knowles. *Self-Directed Learning: A Guide for Learners and Teachers*. Association Press, 1975.

[25] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *Trans. Computing Ed.*, 10(4):1–15, Nov. 2010.

[26] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *Proceedings of WCRE*, pages 214–223. IEEE Computer Society, 2004.

[27] L. E. Margulieux, M. Guzdial, and R. Catrambone. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of ICER*, pages 71–78. ACM, 2012.

[28] S. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example? In *Proceedings of ICSM*, 2012.

[29] M. J. O'Grady. Practical problem-based learning in computing education. *Trans. Computing Education*, 12(3):1–16, July 2012.

[30] Omitted. Blinded for review.

[31] S. Papert. *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc., New York, NY, USA, 1980.

[32] C. Parnin and C. Treude. Measuring API documentation on the web. In *Proceedings of Web2SE Workshop*, pages 25–30, Honolulu, HI, 2011.

[33] J. Piaget. *Origins of intelligence in the child*. Routledge and Kegan Paul, 1936.

[34] E. Soloway, S. L. Jackson, J. Klein, C. Quintana, J. Reed, J. Spitulnik, S. J. Stratford, S. Studer, J. Eng, and N. Scala. Learning theory in practice: case studies of learner-centered design. In *Proceedings of CHI*, pages 189–196. ACM, 1996.

[35] N. Tillmann, M. Moskal, J. de Halleux, M. Fahndrich, and S. Burckhardt. Touchdevelop: app development on mobile devices. In *Proceedings of SIGSOFT/FSE*. ACM, 2012.

[36] V. Tirronen and V. Isomöttönen. On the design of effective learning materials for supporting self-directed learning of programming. In *Proceedings of Koli Calling*, pages 74–82. ACM, 2012.

[37] A. M. Tough. *The Adult's Learning Projects*. The Ontario Institute for Studies in Education, 1971.

[38] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web? In *Proceedings of ICSE*, pages 804 –807, May 2011.

[39] L. Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard University Press, Cambridge, MA, 1978.

[40] A. Zagalsky, O. Barzilay, and A. Yehudai. Example overflow: Using social media for code recommendation. In *Proceedings of RSSE Workshop*, pages 38–42, June 2012.

[41] C. Zander, J. Boustedt, A. Eckerdal, R. McCartney, K. Sanders, J. E. Moström, and L. Thomas. Self-directed learning: stories from industry. In *Proceedings of the Koli Calling*, pages 111–117. ACM, 2012.