Beyond Integrated Development Environments: Adding *Context* to Software Development

Gail C. Murphy

Department of Computer Science
University of British Columbia
Vancouver BC Canada

murphy@cs.ubc.ca

***Abstract*—Software developers create amazing software that is constantly changing the world in which we live: Navigation systems make it easy to find hard to find locations, mobile phones help diagnose health conditions and communication with almost anyone anywhere is virtually effortless. To create these amazing systems, software developers use tooling that is stuck in the past. Integrated development environments enable tools to work together more seamlessly, but remain oriented around the static structure (i.e., files, classes, etc.) of software. This paper proposes that a focus on the study and implementation of *context* could enable software tooling to take a substantial step forward, helping software developers to work more effectively. We delve into initial ideas of what *context* in software development might be and how context might support creating tools that augment human intelligence, allowing developers to better focus on the complex problems they face as they build amazing software.**

***Keywords*-programming, software development environments, productivity**

I. Introduction

When it comes to software, few things have remained as much the same as the tools that software developers use. With these tools, developers create amazing software that is changing the world in which we live: helping us to find the location of friends around the world, communicate with those friends, track our health, and more. For many years, the state of the practice has been to use tools embedded within integrated development environments (IDEs).

These IDEs provide a platform on which tools can be easily built and through which tools can more easily share information and state. The environments in use today are oriented around a platform that exposes the static structure of the software under development. As software developers use static structure to help manage the complexity of systems, this orientation has been beneficial.

In this paper, we refer to the information available to tools to support software development as *context*. Context enables tools to be built that alleviate work for a human trying to perform a task. Consider the content assist tool in the Eclipse IDE[1] that suggests methods that can be called on a variable based on the variable's type. This tool accesses structural information about each available type to save the software developer the need to look up method names and signatures. Context can be thought of as a platform that tools can access, usually through APIs, to get needed information.

The goal of this paper is to motivate the role of context in software development as a first-class construct that enables substantial changes in how researchers approach how software development is supported and performed. We outline a possible future in which context plays a major role in Section II. In this possible future, a developer interacts with a set of tools provided through a smart assistant. By leveraging context, the smart assistant can enable the developer to remain focused on the hard parts of the problems that she is tasked with solving.

Turning this possible future into reality requires taking a broader view to the context needed by tools to support software development activities. In Section III, we outline five different kinds of context: static software structure, dynamic system execution, historical artifact changes, developer activity and team and organization activity. Bringing these kinds of context together in a unified framework requires delving into several new research directions around *recognizing*, *explaining*, *experimenting* with and *building* context. We outline these challenges in Section IV.

This paper will frustrate some readers by staying at the conceptual level. The goal is to reset thinking around software development tools, leaving questions about defining and operationalizing context to future detailed investigations.

II. A Possible Future

Consider a possible future where a rich, evolving context platform is available that enables the creation of tools that alleviate mundane detailed work from a software developer.

A software developer named Jessica comes back to her computer after lunch. An automated assistant (tool) wakes

[1] www.eclipse.org

up and says (writes or presents) to Jessica: "While you were away, a severity two defect was assigned to you that cannot be repaired automatically, you received ten emails from your team about various aspects of projects status, and one of those emails requests that you complete the feature you are working on as soon as possible to enable other work on the team to proceed. What would you like to focus on? (A)

Jessica says back to the automated assistant: "Please show me the description of the severity two defect. The IDE loads the description and Jessica reads through it. Because she has a pretty good idea of what the underlying problem causing the defect may be, Jessica says to the automated assistant: "Load the system configuration causing the defect and focus on code in the module named something like `ChannelIntegrator` (B).

The IDE accesses the right branch of the codebase and says to Jessica, "I have found a module named `ChannelIntegratorStatic` on which the environment is now focused". Jessica begins to work with the code and after some analysis and modifications, asks, "Show me the running times of each of these two methods". The system determines the tests to run that will exercise those methods, builds the system, deploys the system, gathers the execution information, correlates it to the static structure and filters to display the information for the two methods of interest (C).

We use this fragment to illustrate a few points. It is possible to make this scenario real if we are able to implement, and provide access to, an evolving notion of context that enable tools to be developed to help Jessica as she works. Figure 1 sketches a schema of the context at different points in the scenario indicated with capital letters above. These schemas are not complete; they are provided to give a sense of what information would be needed to create the tools necessary to support the scenario. For example, at point A in the scenario, tools would need to be able to access email and the issue repository and resolve individuals named in those artifacts with their role to projects and relationships to Jessica. However, at point B, the context available would need to relate code to issues. We discuss the evolution of context in more detail in Section IV.

Table I outlines for each annotated point the work that is performed by the human versus the tools (noted as the assistant in the table) in the scenario without, and with, context. The table demonstrates that if context can be supported, the developer can remain focused on the task at hand with less of their cognitive effort dedicated to determining *how* to do conceptual steps of the development process, such as the steps required for deployment.

III. Kinds of Context

Based on information needed by tools, we describe at least five different kinds of context that are useful to support software development. For each kind of context, we provide a short description and an example of a tool in which the

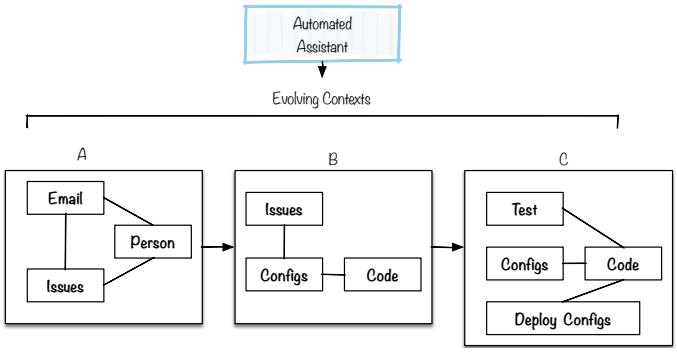

Figure 1. Example of evolving concepts

context is used. We do not claim that these five kinds of context are comprehensive of all kinds of context useful in software development. Instead, we aim to spark a discussion of what other kinds of context do, or should, exist, how we can create tools that use context effectively, and how we can architect different kinds of context to be more accessible at scale within IDEs used in practice.

Static Artifacts. As described in the introduction, state-of-the-practice IDEs provide static source code artifacts as context to tools hosted in the environment. Tools that provide outline views of object-oriented classes or that support search across a source code base make use of this kind of context. Tools have also been built to take advantage of others kinds of static artifact context, such as documentation (e.g., [1]) or question and answer sites (e.g., [2]).

Historical Information: Researchers have also recognized the benefits of tools that access historical information about a system's static artifacts. For example, for a current source code change, the ROSE tool uses historical information about past changes to recommend what other parts of the system might need to change [3]. Although many research tools have been proposed that use historical information, few tools are available to practicing developers. More consideration of how to provide a suitable set of APIs to leverage context as historical information may be needed.

Dynamic Execution: Context in the form of dynamic execution information about a system under development can also enable tools that ease development activities. As one example, WhyLine eases debugging by using system execution information to automatically form questions a developer may have about the execution of the system [4]. These questions take the form of "why did" or "why didn't" something occur to the objects providing the behavior of the system. A common form of dynamic execution information as context in current IDEs is in terms of coverage information describing whether, and how often, different parts of the system's static structure are exercised by tests. Opportunities exist to further leverage dynamic execution context in conjunction with historical information, such as

Table I
COMPARISON OF COGNITIVE LOAD FOR DEVELOPER WITH AND WITHOUT CONTEXT

Annotation	Without *Context*		With *Context*	
	Human	Assistant	Human	Assistant
A	Accesses and cognitively processes multiple email and issues.	N/A	Chooses between defect and feature work	Monitors email and issues, processes and summarizes priority tasks.
B	Determines configuration causing error, accesses version control for correct version, searches for code.	N/A	Waits for code to appear. Cognitive focus remains on task.	Determines configuration causing error, accesses version control for correct version, searches for code.
C	Searches for, and analyzes tests for appropriateness, configures environment to show execution results, finds and runs deploy scripts.	N/A	Waits for performance results to appear. Cognitive focus remains on task.	Searches for, and analyzes tests for appropriateness, configures environment to show execution results, finds and runs deploy scripts.

being able to more easily compare whether the performance of the system is improving or worsening over time.

Individual Developer Activity: Context does not have to be only about what humans produce (i.e., static artifacts) or what happens when the system executes, it can also be about *how* humans work to produce the system. For example, Mylyn builds a degree-of-interest model of the information a developer works with as part of a task and forms a task context based on the frequency and recency of access [5]. Task contexts enable developers to be more productive by making it easy to recall the source code associated with a given task and by enabling other tools, such as content assist, to order information based on work performed as part of the task. Opportunities exist to consider the use of activity over other kinds of artifacts than source code, such as how a developer uses historical information. For instance, context about what branches a developer frequently accesses in a distributed version system might enable the development of recommenders for streamlined workflow.

Team and Organization Activity: Software development is not typically a sole endeavor, instead many complex software systems involve multiple teams working together either within, or across, organizations. Increasingly, industry is focusing on end-to-end management of how software is produced across multiple teams and tools as a value stream [6]. Treating the activities across a value stream as context would enable correlation of downstream effects with upstream choices and would open new opportunities for feedback to be provided to developers as development is undertaken. As a recently emerging kind of context, this kind of context is largely untapped to date.

IV. EVOLVING DYNAMIC CONTEXT

The consideration of context as a first-class construct opens up new opportunities to take a substantial step forward in providing tools for developers that enable the developer to use their cognitive abilities to attack the problems only a human can address. To get to where context can vary in an IDE beyond static software structure, many questions require research.

We need to be able to *recognize* what context is, the forms it takes and to be able to better define and implement it. The ubiquitous computing community has been treating context as a first-class system for almost twenty years. In the ubiquitous computing community, context refers to "information that is part of an application's operating environment and that can be sensed by the application" [7, p.434] and toolkits have been defined to provide access to context, not unlike how IDEs ease access to static artifact structure as context. However, given the kinds of context for software engineering already identified (Section III), the definition of context in software engineering will need to be broader than in ubiquitous computing. Here is one start at a definition: context in software information is *information about the system under development and the environment and process in which the system is being developed*. Future work should consider how context should be defined for software engineering (e.g., the APIs needed) so that researchers can begin to explicitly consider it in their work.

Even without a specific definition of context on which everyone agrees, researchers can begin to explore the idea of context by considering the concept of context and *explain*ing how their work relates to the concept. For instance, imagine a recommender tool that suggests which developer in an organization should be assigned to a particular defect (i.e., [8]). For such an approach, an explicit explanation of the context in which the approach is intended to work could clarify for other researchers, and for those interested in adopting the approach in practice, the environment expected for the approach. To illustrate this point, consider such a recommender: the input of the recommender is a database of resolved defects with information about who solved the defect, the parts of the system the defect is related to and so on; the output is recommendations about whom should be assigned to work on a newly arrived defect. For this recommender, the context is information about which developers in the organization are available to be assigned a defect. This information is **not** required and is thus not an input: a recommendation can be provided without this context, however the likelihood of invalid recommendations—say for a developer on holidays or who has left the organization—rises. The context provides a backdrop to improve the tool.

Once recognized and explained, researchers could be-

gin to more systematically *experiment* with the effects of context for different tools and scenarios. For instance, for a recommender that suggested which command might be executed next by a software developer in an IDE, Gasparic and colleagues considered whether different aspects of the operating environment of the recommender [9], such as the developer and what they were doing, affected the accuracy of the recommender. This experimentation with context is similar to feature sensitivity in machine learning, but places a focus on considering features related to the system and the environment in which it is being developed. An understanding of what elements of context affect different tools can lead to more robust tools for deployment in practice and a better determination of what context needs to be supported to enable tool development.

Research is also needed in how to build the concept of context. With many different kinds of context, is there a need to define an overall ontology of context? How do we architect IDEs of the future to effectively gather and expose context to tools? How can provided context be sensitive to the task and activities of the developer? For instance, must all kinds of context be gathered and exposed when needed or can the gathering of context be triggered by certain actions?

V. Related Efforts

There have been efforts in software engineering that investigate similar notions of context. As mentioned above, Gasparic and colleagues efforts to explore context are outlined above. Bradley and colleagues have considered a context model for supporting conversational developer assistants that consider the context elements needed to support workflow involving a distributed version control system that start to support the assistant concept outlined above [10]. While these efforts explore context explicitly within a particular instance, they do not call out the need to consider context as a first-class construct, which is the purpose of this paper.

Others have also been exploring what is next beyond the current state-of-the-practice IDEs. Code Bubbles redefines the interface of the IDE around editable fragments [11]. Multiple bubbles can be visible at once, helping developers to see and interact with working sets important to the tasks they are performing. Light Table takes code bubbles another step forward by integrating executions of the system under development into the IDE to allow interrogation.[2] An explicit consideration of the different kinds of context in light of these approaches may yield new opportunities for tools to help developers.

VI. Summary

As consumers, we are seeing an increasing number of tools introduced to ease our access to information. For instance, Alexa enables a human to ask a question about the weather today and obtain an answer without finding a web browser, locating a weather page and reading the result themselves. When computerized assistants can access context, such as where we are and who we are, they can provide humans the ability to focus on the the task at hand, instead of the mechanics of performing the task.

In this paper, we have argued that explicit attention and investigation of *context* as a first-class construct in software engineering enables a leap forward in how tools support software developers, much like the leap forward being taken with tools like Siri. A focus on context can surface the many different kinds of information useful to tools, such as historical and activity information, and can enable the provision of such information to allow for the creation of tools that truly augment human intelligence. It is time for us to move beyond the limited notion of context available in current development environments.

[2] www.chris-granger.com/2012/04/12/light-table-a-new-ide-concept

Acknowledgment

This work was supported by an NSERC Discovery Grant.

References

[1] B. Dagenais and M. P. Robillard, "Recovering traceability links between an API and its learning resources," in *34th Int'l Conf. on SE*, 2012, pp. 47–57.

[2] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza, "Prompter: A self-confident recommender system," in *30th Int'l Conf. on Soft. Maint. and Evol.*, 2014, pp. 577–580.

[3] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *26th Int'l Conf. on SE*, 2004, pp. 563–572.

[4] A. J. Ko and B. A. Myers, "Designing the whyline: a debugging interface for asking questions about program behavior," in *2004 Conf. on Human Factors in Comp. Sys.*, 2004, pp. 151–158.

[5] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *14th ACM SIGSOFT Int'l Symp. on Found. of SE*, 2006, pp. 1–11.

[6] M. Kersten, *Project to product: How to survive and thrive in the age of digital disruption with the flow framework*. IT Revolution, 2018.

[7] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: Aiding the development of context-enabled applications," in *Conf. on Human Factors in Comp. Sys.*, 1999, pp. 434–441.

[8] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *28th Int'l Conf. on SE*, 2006, pp. 361–370.

[9] M. Gasparic, G. C. Murphy, and F. Ricci, "A context model for ide-based recommendation systems," *Journal of Sys. and Soft.*, vol. 128, pp. 200–219, 2017.

[10] N. C. Bradley, T. Fritz, and R. Holmes, "Context-aware conversational developer assistants," in *40th Int'l Conf. on SE*, 2018, pp. 993–1003.

[11] S. P. Reiss, J. N. Bott, and J. J. LaViola. Jr., "Code bubbles: A practical working-set programming environment," in *34th Int'l Conf. on SE*, 2012, pp. 1411–1414.