Transferring Code-Clone Detection and Analysis to Practice

Yingnong Dang[†], Dongmei Zhang[*], Song Ge[*], Ray Huang[*], Chengyun Chu[†] and Tao Xie[‡]
[*]Microsoft Research Asia, China
Email: {dongmeiz;songge;rayhuang}@microsoft.com
[†]Microsoft Corporation, USA
Email: {yidang,chchu}@microsoft.com
[‡]University of Illinois at Urbana-Champaign, USA
Email: taoxie@illinois.edu

***Abstract*—During software development, code clones are commonly produced, in the form of a number of the same or similar code fragments spreading within one or many large code bases. Numerous research projects have been carried out on empirical studies or tool support for detecting or analyzing code clones. However, in practice, few such research projects have resulted in substantial industry adoption. In this paper, we report our experiences of transferring XIAO, a code-clone detection and analysis approach and its supporting tool, to broad industrial practices: (1) shipped in Visual Studio 2012, a widely used industrial IDE; (2) deployed and intensively used at the Microsoft Security Response Center. According to our experiences, technology transfer is a rather complicated journey that needs significant efforts from both the technical aspect and social aspect. From the technical aspect, significant efforts are needed to adapt a research prototype to a product-quality tool that addresses the needs of real scenarios, to be integrated into a mainstream product or development process. From the social aspect, there are strong needs to interact with practitioners to identify killer scenarios in industrial settings, figure out the gap between a research prototype and a tool fitting the needs of real scenarios, to understand the requirements of releasing with a mainstream product, being integrated into a development process, understanding their release cadence, etc.**

I. Introduction

During software development, software engineers commonly reuse a code fragment by copying and pasting it from another location of the same code base or even a different code base. Such reused code fragments may or may not undergo modifications or adaptations after it is pasted in the new code location. These same or similar reused code fragments are named as code clones. Detecting and analyzing code clones in code bases have been shown to be useful towards various software-engineering tasks such as bug detection and refactoring. For example, code clones may be candidates for refactoring (e.g., refactoring multiple copies of a code clone into a single location) [1]. Inconsistencies across multiple copies of a code clone may indicate buggy code locations to inspect and fix [2], [3]. In addition, given a known buggy or vulnerable code fragment, clone search can be used to detect other clone copies as potentially buggy or vulnerable code fragments that need to be inspected and fixed [4].

To explore research topics in the area of code clones, numerous research projects [5] have been carried out, especially on empirical studies or tool support for detecting or analyzing code clones. However, very few of these research projects were conducted in industrial contexts, and much fewer resulting research tools were actually used or adopted by industrial practitioners. The research community has already realized gaps between academic research and industrial practices [6]–[8], and has called for training and education of researchers and practitioners in conducting successful technology transfer and adoption.

In this paper, we report the successful technology-transfer case of XIAO [9] and lessons learned from our research efforts. In particular, XIAO has been shipped as part of a popularly used industrial IDE Visual Studio, and it has been adopted in the Microsoft Security Response Center. Visual Studio is Microsoft's flagship product for developer productivity. Millions of developers around the world are using Visual Studio for their development work. Therefore, as part of Visual Studio, XIAO is benefiting a large developer community.

The Microsoft Security Response Center (MSRC) is responsible for the security issues of Microsoft software products. The engineers of MSRC conduct an investigation process for each security vulnerability. As an important step of this process, MSRC engineers must figure out whether there are similar vulnerabilities in all other Microsoft software products. Before XIAO was transferred, such needs were fulfilled by manual email communication between MSRC and different product teams. Such manual email communication was a highly time-consuming process, and it was difficult to provide high-confidence assurance that no or few similar vulnerabilities were missed. In 2009, an online clone-search service provided by XIAO was deployed and used by MSRC engineers to help with their investigation on security vulnerabilities. Since then, the productivity of MSRC engineers has been significantly improved.

According to our experiences of transferring XIAO, technology transfer is a rather complicated journey that needs significant efforts from both the technical aspect and social aspect. From the technical aspect, significant efforts are needed to adapt a research prototype to a product-quality tool that addresses the needs of real scenarios, to be integrated into a mainstream product or development process. From the social

aspect, there are strong needs to interact with practitioners to identify killer scenarios in industrial settings, figure out the gap between a research prototype and a tool fitting the needs of real scenarios, to understand the requirements of releasing with a mainstream product, being integrated into a development process, understanding their release cadence, etc.

The rest of the paper is organized as follows. Section II presents the target task scenarios of XIAO. Section III illustrates the overview of XIAO. Section IV presents the industry impacts of XIAO. Section V discusses the lessons learned during the technology transfer of XIAO. Section VI discusses and compares our experiences to the related experiences reported in the literature, and Section VII concludes the paper.

II. Target Task Scenarios

At the very beginning of the XIAO project, we kept in mind that we needed to generate actionable results for developers in terms of detected clones. We started the project from understanding the important scenarios that developers may want to devote their time to. Based on our investigation, the first scenario is *fix bug once*. The reason is that code bugs spread via code duplication. In particular, if code bugs are security vulnerabilities, the benefit of fixing the bugs at all cloned copies is huge: helping secure the software products. The second one is *code refactoring*. Especially in a large enterprise, there are two big motivations that developers want to reduce code duplication: (1) reduce the overall footprint of the code base; such reduction is especially important for some memory-critical components such as kernel components; (2) prevent code bloating and reduce the maintenance cost; such prevention/reduction is especially important for a software project with long history, e.g., 10 years or longer, which is typical for many enterprise software systems. The accumulation of development time makes the duplication situation more severe along the way and leads to the increase of maintenance difficulty.

III. Overview of XIAO

XIAO has four main characteristics [9]: high tunability, scalability, compatibility, and explorability.

High tunability of XIAO is achieved with a new set of similarity metrics in XIAO, reflecting What You Tune Is What You Get (WYTIWYG): users can intuitively relate tool-parameter values with the tool outputs, and easily tune tool-parameter values to produce what the users want. For example, the similarity-parameter value of 100% should lead to outputs of two exactly same cloned snippets, and the 80% value should lead to outputs of two cloned snippets with 80% similarity judged by the users. The parameters of the proposed metrics in XIAO enable users to effectively control the degree of the syntactic difference between the two code snippets of a near-miss clone pair: the degree of the statement similarity, the percentage of inserted/deleted/modified statements in the clone pair, the balance between the code-structure similarity, and the quantity of disordered statements. Such high tunability of XIAO is critical in applying an approach of code-clone detection such as XIAO to a broad scope of software-engineering tasks such as refactoring and bug detection since these different tasks would require different levels of parameter values. For example, when using XIAO to search code clones for detecting similar security vulnerabilities, engineers need high recall of the detection results to make sure that all or most potential security vulnerabilities should be detected, while not very high precision can be tolerated. In contrast, when using XIAO to search code clones for identifying refactoring candidates, engineers want to prioritize which code clones to work on first such that their invested efforts can bring the greatest benefit on reducing footprints via refactoring.

High scalability of XIAO in analyzing over ten million lines of code is achieved with a well-designed scalable and parallelizable algorithm with four steps. These four steps include preprocessing, coarse matching, fine matching, and pruning. Preprocessing transforms source-code information to filter out inessential information such as code comments, and map code entities such as keywords and identifiers to tokens. Such information preprocessing reduces the cost of the actual analysis. To offer high scalability, XIAO splits the main analysis into two steps: coarse matching and fine matching. Coarse matching is less costly but less accurate than fine matching. The scope narrowed down by coarse matching is fed to fine matching, achieving a good balance on analysis scalability and accuracy. The step of pruning further improves the analysis accuracy. In addition, the clone-detection algorithm of XIAO can be easily parallelized. XIAO partitions the code base and executes multiple instances of clone detectors simultaneously. Each instance detects clones on a number of pairs. The results of all the instances are then merged.

High compatibility of XIAO in analyzing code in different development environments (such as different build systems) is achieved with its compiler-independent, lightweight, and pluggable parsers. XIAO has built-in parsers for the C/C++ and C# languages. We define an open Application Programming Interface that allows the easy plug-in of parsers to support various programming languages. Note that the parsing task is much lighter than the comprehensive functionalities offered by compilers. Compared to the approaches of parse-tree-based clone detection such as DECKARD [10], our approach has the advantage of compiler independence; it can be easily applied to accommodating different language variants and build environments, which typically exist in real settings of software development, especially for C/C++ [11].

High explorability of XIAO allows users to easily explore and manipulate the detected code clones. Such high explorability is achieved with its carefully-designed user interfaces including visualization support. We design a simple heuristic to define the level of difference between cloned snippets. We also use the metric to rank clones to prioritize the review of clones to identify bugs. XIAO includes clone visualization to clearly show the matching blocks and the block types of a clone pair. In this way, users can quickly capture whether there

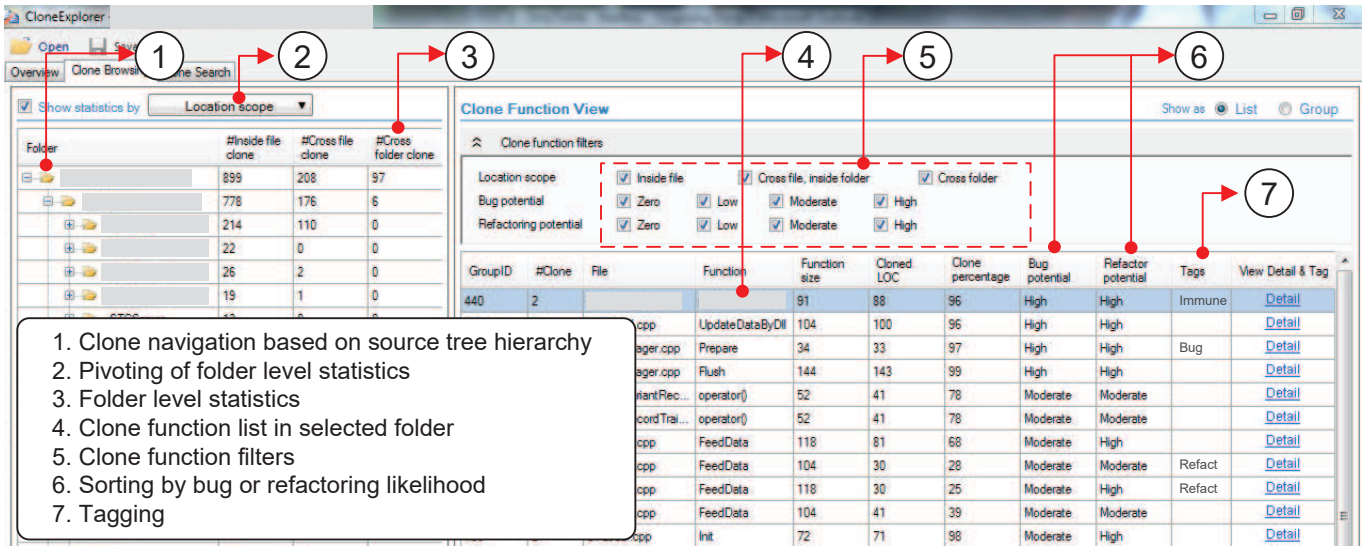

Fig. 1. User interface of XIAO's Clone Explorer

is any difference between the two cloned snippets, what kind of difference it is, and how much difference there is. XIAO also includes a tagging mechanism to help coordinate joint efforts of reviewing code clones from multiple engineers.

Figure 1 shows the user interface of XIAO's Clone Explorer. It organizes clone statistics based on the directory hierarchy of source files in order to enable quick and easy review at different source levels (Figure 1-(1)). A drop-down list (Figure 1-(2)) is provided to allow pivoting the clone-analysis results around the bug likelihood (Figure 1-(3)), refactoring likelihood, and clone scope. Clone scope indicates whether cloned snippets are detected inside a file, cross-file, or cross-folder. For a selected folder in the left pane, the right pane (Figure 1-(4)) displays the list of clone functions (those functions including cloned snippets), which could be sorted based on bug likelihood or refactoring likelihood (Figure 1-(6)). Filters (Figure 1-(5)) on the clone scope, bug likelihood, or refactoring likelihood are provided to enable easy selection of clones of interest.

IV. Industry Impacts

The industry impacts produced by the XIAO project are three-fold [9]: Microsoft product development, Visual Studio shipping, and vulnerability investigation at the Microsoft Security Response Center.

Being adopted for Microsoft product development. During the initial phases of the project, we released XIAO inside Microsoft for different development teams to use (with the first version released in April 2009). There were more than 750 downloads of the tool as of the end of year 2010. The sizes of code bases that XIAO is used to detect code clones vary from several thousand lines to over 50 million. Engineers from different teams in Microsoft have used XIAO in their development process. For example, one developer from an application software development team used XIAO to identify potential bugs caused by inconsistent code clones and to find refactoring candidates. He reviewed 69 clone groups with 180 cloned snippets in total and found that about 10% have potential code bugs caused by inconsistent code clones, and 33% of them could be refactored. Another developer used XIAO to reduce the footprint of a component with more than 200K LOC. As a result, the percentage of cloned LOC was reduced from 30% to 25%. A testing team developed a plug-in parser to parse the language used in their product and used the tool to find duplicated test code and then refactor it.

Being shipped with Visual Studio. Later XIAO was further transferred as the feature of code clone analysis shipped in Visual Studio. Initially, XIAO was developed and released only as a plug-in of Visual Studio (not being part of a Visual Studio release) so that Visual Studio users could easily download and install XIAO, and use it with Visual Studio in their daily work. However, much higher industry impact could be achieved when XIAO was shipped with Visual Studio (i.e., as an inherent part of Visual Studio releases) for two main reasons. First, the process for delivering XIAO to Visual Studio users is expedited. After being shipped with Visual Studio, the XIAO feature is readily available once a Visual Studio user has the Visual Studio package installed. There is no time delay from the time when the new Visual Studio package is installed to the time when the Visual Studio plug-in (specially developed for this new Visual Studio package) is downloaded and installed. Second, the readiness and convenience to Visual

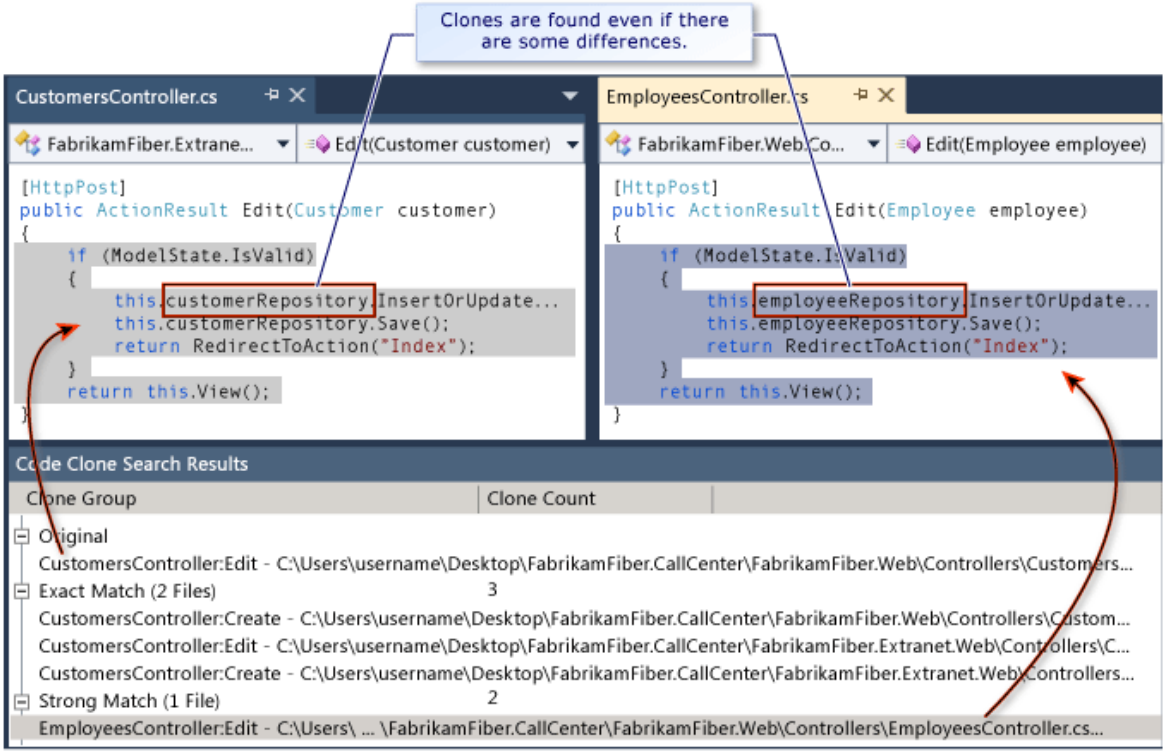

Fig. 2. User interface of clone analysis (based on XIAO) in Visual Studio

Studio users are naturally provided. After XIAO has been shipped with Visual Studio, there is no need for the Visual Studio users to download a XIAO plug-in, and install it. More importantly, the XIAO feature shipped with Visual Studio just works: the Visual Studio users do not need to worry about the compatibility issue of the XIAO plug-in with the new Visual Studio package or other Visual Studio plug-ins.

Figure 2 shows the user interface of clone analysis (based on XIAO) in Visual Studio. The feature of code clone analysis in Visual Studio can help developers efficiently identify code-refactoring opportunities and quickly find bugs in duplicated code. During the release of a new version of Visual Studio, highlights on the feature of code clone analysis presented by high-profile Microsoft managers indirectly reflected the importance of the XIAO feature in the release. For example, before the release, in the TechEd North America 2011, Cameron Skinner, General Manager of Visual Studio Ultimate at Microsoft, gave a talk on "the Future of Microsoft Visual Studio Application Lifecycle Management", highlighting the feature of code clone analysis (at the 26'-29' time of http://channel9.msdn.com/Events/TechEd/NorthAmerica/2011/FDN03). After the release, at BUILD conference 2011, Jason Zander, a Corporate Vice President of Visual Studio demonstrated the feature of code clone analysis in his keynote talk (at 12'30" time of http://channel9.msdn.com/events/BUILD/BUILD2011/KEY-0002).

A broad scope of Visual Studio users have found the feature of code clone analysis useful. Via the Visual Studio UserVoice site (a website for gathering what Visual Studio users would like to see in future versions of Visual Studio), some users suggested Microsoft to "allow code clone detection to be run from command line and/or MSBuild", gathering 139 support votes. The reason is that the clone analysis feature in the Visual Studio release limits the scope of clone detection to be the currently opened solution (i.e., a container for projects to track dependencies among projects) in Visual Studio. Such limit cannot enable many users to run code-clone detection against large code bases that have a large number of solutions.

The following are examples of user comments gathered from broad Visual Studio users (outside of Microsoft) in online discussion.

- "Code clone is very interesting feature for a code review. It would be extremely useful if we can export analysis results in human readable format. Our outsources usually have no ultimate version of visual studio and its hard for them to fix all issues of code cloning."
- "In order to make a code coverage analysis, for our projects, it's also required to make a code clone analysis. I know that it's possible on Visual Studio IDE, but for Agile projects, it's more powerful by command line."

Note that currently Visual Studio does not ship a command-line version to external users. But the internal tool has a command-line version, being used by many internal users.

Being adopted at the Microsoft Security Response Center (MSRC). Since May 2009, an online clone-search service provided by XIAO has already been integrated into the engineering process of MSRC, as a step of identifying similar vulnerabilities across Microsoft products. Specifically,

the code snippet of an identified vulnerability is automatically sent to the online clone-search service, and the returned code snippets are sent to MSRC engineers for them to verify whether there is real security vulnerability. In the Microsoft Security and Defense blog, MSRC publicly acknowledged the usage and benefit of using XIAO. In a blog entry[1] posted in May 2012, an MSRC engineer states: "we wanted to be sure to address the vulnerable code wherever it appeared across the Microsoft code base. To that end, we have been working with Microsoft Research to develop a 'Cloned Code Detection' system that we can run for every MSRC case to find any instance of the vulnerable code in any shipping product. This system is the one that found several of the copies of CVE-2011-3402 that we are now addressing with MS12-034."

During the initial period of using XIAO at MSRC, there were more than 590 million lines of code being indexed. During the second half of year 2010, there were a number of vulnerable code snippets searched against the XIAO service. Among these search cases, 18.3% cases of them found good hits, i.e., for these cases, the security-engineering team needs to do further investigation to confirm whether there are duplicated vulnerabilities. Given high severity of security bugs, 18.3% good-hit cases are very good results.

As an example, in one of the MSRC cases, a reported security vulnerability could cause potential heap corruption and lead to remote code execution. After investigation, the vulnerable code snippet was found in code base A: a buffer-overflow check was missing. Using XIAO's clone-search service, one security engineer on the security-engineering team found three clones of the vulnerable code snippet - one is also in code base A and the other two belong to code base B. This security engineer contacted the code owners of these three cloned snippets and confirmed that one snippet in code base B was indeed vulnerable. After the contact, the development team owning the vulnerable cloned snippet in code base B had confirmed to fix this security bug while the security bug in code base A was fixed.

XIAO's clone-search service has greatly improved the productivity of the security engineers and it enhanced the reliability of the bug-investigation process as well. Based on the clone-search results, security engineers are able to obtain a better understanding of the potential impact of security vulnerabilities, as well as communicating more effectively with development teams on vulnerability investigation and fixing.

V. Lessons Learned

In this section, we next share some of our experiences and the lessons learned during the process of transferring XIAO to broad industrial practices: being shipped as part of Microsoft Visual Studio and being deployed and intensively used in daily work at the Microsoft Security Response Center.

[1]http://blogs.technet.com/b/srd/archive/2012/05/08/ms12-034-duqu-ten-cve-s-and-removing-keyboard-layout-file-attack-surface.aspx

To facilitate the discussion of the experiences and lessons in this section, we present four stages of transferring a tool such as XIAO from the technical and social aspects in Table I. We classify the first seven lessons learned into these two aspects: the technical aspect (the first three from Sections V-A to V-C) and the social aspect (the next four from Sections V-D to V-G). The lessons learned from the technical aspect are especially important in Stage 2 (from research prototype to early industrial adoption), to make the technology mature enough for integration into a main stream product or key engineering process, and convince the key decision maker to kick off the productization. The lessons learned from the social aspect are important for both Stages 2-3, because in both stages we need comprehensive collaboration and engagement with product teams and early users. Finally, through the whole transfer process, we keep in mind to identify more scenarios that the same technology can be applied to, as illustrated in the last lesson learned (Section V-H).

A. Focusing on Problems Well Recognized in Practice

There are two models (or their mixture) of initiating a research project in the context of technology transfer, as reported in the literature [12]: the *pull* model and *push* model. For example, initiating the StackMine project [13] primarily followed the pull model. Before the StackMine project was started, during one of the meetings with a research team, a member from a product team talked about their state of practice in inspecting a single stream of call-stack traces for performance analysis, as well as the challenges that they were facing on inspecting a large number of trace streams. Then the research team started the StackMine project to address the (most urgent) need of the target customers. Typically, projects of this pull model may more easily fit in the workflow of the target product team, facilitating the integration of the resulting tools in the product team's activities.

In contrast, initiating the XIAO project primarily followed the push model. Based on the research literature and initial investigations of some Microsoft code bases, our research group gained insights and realized the existence of bugs related to code clones especially near-miss clones, but did not know their extent. Then our research group developed the XIAO prototype and demonstrated the prototype to various Microsoft product teams to "sell" the solution to them. Through iterations of interactions with product teams, our research group concretized the details of the target tasks of refactoring and bug detection. Furthermore, through iterations, the group also understood that potential customers may be more willing to carry out refactoring during early stages of the customers' software development project because in later stages, the customers may be more willing to fix bugs rather than refactoring. Such understanding helps successfully acquire customers by selling different tasks depending on the potential customers' stages.

Different product teams (not necessarily all or most product teams) in Microsoft may have their own respective incentives for adopting and benefiting from XIAO. It was important for

TABLE I
STAGES OF TRANSFERRING A TOOL FROM THE TECHNICAL AND SOCIAL ASPECTS.

Stage	Technical Aspect	Social Aspect
1. From research idea to research prototype	• Sound algorithms and research contributions are validated in lab settings. • Scalability, reliability, compatibility, usability issues may be only partially addressed. • The prototype can well demonstrate its unique value in specific settings, but may not well address real-scenario requirements.	• Researchers are the main driver, with fair or good visibility in academia (e.g., via academic publications) but may have little visibility in industry to attract industrial interest.
2. From research prototype to early industrial adoption	• Killer scenarios in practice are identified and well recognized. • Additional research on algorithm improvement is conducted to fit the needs in the killer scenarios. • The prototype is adapted/packaged to be a tool that fits the needs of killer scenarios in industrial settings. • Most scalability, reliability, and compatibility issues are well addressed. • Gaps and fixing strategies on integration with a mainstream product and key engineering process are identified.	• Researchers and practitioners reach mutual agreement and co-drive the process. • Valuable feedback is obtained from practitioners on key scenarios and requirements on scalability, reliability, compatibility, and usability. • A sufficient number of early adopters are attracted in industry. • Key decision makers in industry decide to integrate the tool into a mainstream product or key engineering process.
3. From early industrial adoption to broad industrial impact	• Technical issues (e.g., performance, interface, compatibility) and process issues (e.g., workflow, UI) are fully addressed.	• Practitioners drive the process, while researchers play a supporting role. • The tool is released as part of the mainstream product, or integrated into the key engineering process.
4. Post-transfer	• Additional scenarios in real industrial settings that the key technology can help may be identified, and new transfer opportunities are identified.	• Post-release feedback and further feature requirements are attained from a broad user base, and a future release plan is made as needed.

us to identify such product teams and their incentives early on and take efforts to convert them to be pilot users (i.e., early adopters) of XIAO. Below we list those major product teams in Microsoft that had strong incentives to adopt XIAO.

OS. While interacting with product teams working on OS, our research group found that the product teams typically wanted to reduce the whole footprint of the core of OS. Then refactoring code clones could be highly beneficial there.

Office. The Office division has multiple teams with a rich set of shared functionalities. In addition to a set of shared libraries that are being used across teams, there is a fair number of functionalities that have multiple copies with variance being used across the teams, due to the higher cost of abstraction, e.g., dependencies across the teams, rather than duplication. It is important to provide better management of such cloned copies, especially ensuring that bugs found in a copy of such code is fixed in all cloned copies. An engineer from Office product teams actively outreached to our research group for dogfood (i.e., internal trial out) right after he saw the demo of XIAO.

Dynamics (a Customer Relationship Management product). A non-object-oriented scripting language for software development is widely used across the Dynamics teams. The scripts in this language are difficult to be abstracted as modules and therefore cloned copies are spreading across multiple similar versions of a Dynamics product for many customers in different vertical domains. Making common changes across these different vertical domains is time consuming and error prone. Providing a parser interface to these product teams could enable XIAO to be able to detect clones in their code written in the scripting language, and thus the product teams can easily figure out all clone copies across vertical domains that need to undergo a specific change. Based on the parser interface provided by our research group, the Dynamics teams developed their own parser to parse the scripts.

Microsoft Security Response Center (MSRC). The MSRC team is responsible of addressing security issues across Microsoft. A big challenge of the team is to quickly identify all cloned copies of a piece of vulnerable code found in one product, across all code bases of important Microsoft products. The reason is that in many cases a library/package is copied across multiple products.

B. Supporting Human Interactions with Tool Configurations and Results

As discussed earlier, high tunability of XIAO is critical to allow XIAO to be applied in a variety of application scenarios. Users need to intuitively relate tool-parameter values with the tool outputs, and easily tune tool-parameter values to produce what the users want. For example, in the application scenario of investigating security vulnerabilities, security engineers would like to have high recall of clone detection (i.e., little chance of missing clone copies). Therefore, XIAO has its default similarity threshold value of 0.6, a relatively small value. The value is tunable by security engineers to achieve even higher recall.

Supporting human interactions with the tool results (e.g., detected clone copies) is also very important. We received a lot of feedback on the usability of XIAO, and in particular the XIAO GUI for code-clone exploration and visualization. We released

several versions of the tool, mainly focusing on improving the usability based on the user feedback. First, the XIAO GUI needs to provide efficient exploration. There can be a great number of clones found in large-scale code bases. Therefore, it is important to enable engineers to explore the detected clones easily and efficiently. According to our discussion with and observation of engineers, the engineers have a need to sort and filter clones based on different metrics, e.g., source tree structure, size of cloned snippets, size of a clone group, size of difference between different copies, etc. Second, the XIAO GUI needs to provide intuitive visualization of differences across clone copies. The types of differences between clone copies can be diverse; engineers want to identify the most important differences first. Doing so can help the engineers quickly find out whether there are any potential bugs in the cloned code or whether the cloned code needs to be refactored. XIAO classifies the clone differences into four categories and uses visualization techniques to enable users to view the near-miss code clones intuitively. Third, the XIAO GUI needs to provide a tagging mechanism to allow users to keep track of their inspection results across versions of their code (instead of re-inspecting the same subset of clone detection results across the versions). Not all detected clones are of interest to the engineers. For example, some clones are by-design and they should not be refactored. It is difficult to automatically filter out the by-design clones because deep domain knowledge may be required. XIAO provides a tagging functionality for engineers to tag code clones in three categories: immune (uninteresting), problematic inconsistencies, and refactoring opportunities.

C. Addressing Tool Efficiency and Robustness

We took significant efforts to make XIAO efficient to run. Based on direct interactions with internal XIAO users at Microsoft, these users put such tool efficiency as their high-priority requirement. Insufficient tool efficiency was also a blocking factor for Visual Studio integration. According to our discussion with engineers, typical expectations were that detecting clones from 1 million lines of code should be within 5 minutes in a common commodity PC. Clone detection should be still efficient when there exist extreme cases such as a function with 10K+ lines of code. In some production code bases, there are extremely long functions for hosting a language model. These functions are machine-generated code. However, XIAO might not have such knowledge to exclude such functions.

Having good scalability is critical in application scenarios of XIAO. First, when investigating security vulnerabilities, a vulnerable code segment needs to be used as query to search through hundreds of millions lines of code from multiple code bases. Second, an individual production code base can easily reach tens of millions lines of code. The development teams of a software product often have the need to get an overall picture of cloning situation of their product code base to assess the code refactoring efforts.

High tool efficiency and robustness are especially important for integration and shipping with Visual Studio. There were one researcher and one developer (from our research group) working together on the original research-prototype development and internal tool release at Microsoft. There were two dedicated developers from the tech-transfer engineering team of our research lab for working closely with us on consolidating our research-prototype-quality code to production code, refactoring the structure to fit the needs of integration into Visual Studio user experience. These two developers identified a number of corner cases that our original algorithm did not handle well or did not perform well. One example is that our original algorithm did not perform well when detecting clones inside one function. Another example is that we were in need of an elegant way to handle short functions. Our original algorithm just used a simple threshold on the minimum number of lines of code to exclude short functions. But in a clone-search scenario of Visual Studio integration, testers from the Visual Studio team found that it was important that our tool needed to return clone results within even very short functions (e.g., with only 3 to 5 lines) when users search these functions. The reason is that even when a function is very short, it is possibly cloned and with code bugs spreading in multiple places. To make algorithm improvement, these two developers from the tech-transfer engineering team needed to get very familiar with many details of our original algorithm.

During the process of transferring XIAO to the Visual Studio team, one special challenge that we faced was the loading performance. At the beginning, we hooked up our parser and indexing component when Visual Studio loaded a new solution to memory. However, our component led to a longer solution-loading time, which is not acceptable from the overall Visual Studio experience: very simple operations on triggering detection and simple exploration UI to consume results. One special case considered by us was that there is often machine-generated code (UI solution) in a project, users typically do not care clone results on such machine-generated code since such results are not actionable. We needed to automatically filter out such source files. We needed to balance on UI simplicity and flexibility on excluding some source files from detection. Our final solution was to use an XML configuration file to allow users to filter out source files. Such functionality is just for advanced users so that it does not appear in the UI layer. However, if users add an XML configuration, our tool can recognize it. We had to change it to a later stage when users click the "clone detection" button to start parsing and indexing. Another case was that during Visual Studio integration, we needed to switch from our own parser to the Visual Studio language parser. However, we encountered a memory-leak bug when using the Visual Studio language parser. The collaborating product team did not own the parser. Addressing such issue requires cross-team coordination.

D. Having "Insider" Champion

To transfer XIAO to the Microsoft Security Response Center (MSRC) or Visual Studio, having an "insider" champion (as a

true believer of the technology or tool being transferred) in the partnering product team is an important success factor. In the case of transferring XIAO to MSRC, the fifth author of this paper, being the sponsor from MSRC, played an important role in this transfer in three main aspects. First, the MSRC sponsor convinced his management to support his collaboration with our research group; in many cases, a product team often does not fully trust the quality of a tool prototype developed by a research team. Second, the MSRC sponsor conducted necessary integration work for making the clone search as part of the investigation process for security vulnerabilities, i.e., for each incoming case of security vulnerability, an automatic query against the clone-search service is triggered, and then the returned clone-search results are organized in their format, and embedded in the overall organization of the documentation archive for the security-case investigation. Third, the MSRC sponsor also needed to operate/maintain the clone-search service that we deployed for them. The MSRC sponsor needed to collect all code bases to be indexed, find machines to host the service, install the service, etc. Fourth, the MSRC sponsor tested the service for their MSRC scenarios. Fifth, the MSRC sponsor educated the MSRC teammates to use the service, remind/push security analysts to review the returned result from clone search at the beginning. Note that after the team had a couple of successful use cases (i.e., clone search helped them identify cloned vulnerabilities), the MSRC sponsor did not need to remind his MSRC teammates again.

Similarly, in the case of transferring XIAO to be shipped with Visual Studio, a manager from the Visual Studio team served as such role of "insider champion". This manager played three major roles. First, the manager raised the awareness (to the Visual Studio team) of the quality of XIAO, from various levels of managers to individual developers. Second, the manager raised the awareness (to our research group) of the product-development timing, e.g., when the team's product-planning stage starts, what product-plan requirements are. Third, the manager connected our research group with corresponding feature teams to solve integration issues, such as hook-up points in Visual Studio UI, how to develop Visual Studio plug-in packages.

E. Engaging with Partnering Product Team Veridically

Deep engagement and well alignment with the Visual Studio team is critical for transferring XIAO to be shipped with Visual Studio. Our research group took significant efforts communicating with relevant people at different levels and different disciplines in the Visual Studio team. For example, our research group communicated with the Director of Program Managers to understand the overall release schedule of Visual Studio and how the feature of code clone analysis can fit in, and asked for the director's guidance on how to efficiently work with the program managers to ensure the alignment between our research group and the Visual Studio product team. Our research group also communicated with a Principal Development Manager on the detailed Visual Studio development schedule and how the two collaborating parties can work together on the transfer of XIAO. In addition to reaching out to the middle and high level managers, our research group also closely engaged with individual developers and testers from the Visual Studio team to make sure that they understood what should be done for this transfer and the technical details that they should pay attention to in the integration. Our research group also paid attention to get support from the executive of the product team to ensure smooth technology transfer. For example, our research-group manager communicated with the general manager a couple of times to understand the overall context of Visual Studio ship wheel (i.e., the cadence of a product's shipment) and ensure that there were no potential issues for this technology transfer.

F. Building Iterative Feedback Loop with Partnering Product Team

Proactively seeking quick feedback from the partnering product team helped our research group better identify real problems and correctly formulate problems, raise the group's confidence on the usefulness of the technology to be transferred, and enable the group to build a tool with wide coverage of various important application scenarios.

In particular, our research group proactively sought feedback for ensuring technology readiness and User Interface (UX) understanding before the transfer. At the very early stage, our research group had actively sought feedback on the research prototype of code-clone analysis from multiple product teams, including Visual Studio. This step greatly helped the group to better understand user scenarios, user experiences, and ensure technology readiness. For example, our research group asked a member from a specific product team to help review results of clone detection for their code base. This step helped the group get first-hand knowledge about how useful the technology could be to developers. This product team member also gave our research group a lot of comments on the UX feature requirements. Our research group also contacted a Distinguished Engineer of a product team to use code-clone detection to refactor their code. The Distinguished Engineer also gave the group very insightful comments. The early engagement with an Visual Studio program manager (later becoming the "insider" champion as mentioned in the preceding section) was also an important step toward the actual technology transfer. Our research group worked with the program manager for about six months to drill down to the details about the UX of code-clone detection for Visual Studio users, to build a Visual Studio Add-in to explore the end-user experience and technology feasibility. With this collaborative effort, the program manager was convinced on the potential value of the technology to Visual Studio users. Consequently, the program manager helped the group conduct a successful General Manager review at the Visual Studio planning time frame, leading to the official kick-off of the technology transfer on code-clone detection.

G. Conducting Efficient Joint-Development Process

The members of the joint-development team include people from the tech-transfer engineering team in our research lab and our research group. These members closely collaborated with the Visual Studio team, and actively sought feedback from engineers in multiple product teams. Efficient joint-development process was important to ensure successful execution of the XIAO transfer. Our research group came up with an efficient mechanism to work with the Visual Studio team led by a development manager and program manager. The two collaborating parties had a weekly sync-up meeting to define backlogs, triage bugs, and discuss technical issues. For urgent and important issues, relevant team members gave heads-up to the development manager and program manager so that they could investigate these issues before the sync-up meeting, to better help the two collaborating parties to solve the issues in the meeting. Our research group also closely collaborated with the product team on solving difficult technical issues. For example, the feature of code-clone analysis needs to consume some CPU time at Visual Studio's initialization stage, which increased the overall launch time of Visual Studio. Two team members gave a number of suggestions to mitigate this issue. These suggestions helped the product team construct the final solution on solving the issue. Another example is a memory-leak issue found in a native library. At the beginning, the memory leak occurred only when the feature of code-clone analysis was enabled. Therefore, two team members took a lot of efforts to debug the issue and finally identified the root cause. By communicating the result with the product team, our research group finally got support from the library owner and resolved the issue.

H. Proactively Expanding Applicable Scope

To identify new transfer opportunities for the XIAO project, in later phases of the project, we expanded the applicable scope of the XIAO algorithm to conduct research on boosting productivity of software development specifically from a change-centric perspective. The reason is that change is a very basic concept and artifact during the entire software development process. A significant part of software development efforts can be viewed as change oriented: from change requirement to making changes, from reviewing changes to testing changes.

In particular, we expanded the applicable scope of the XIAO algorithm from clone detection or searching to change understanding, with focus on (1) change visualization for code review, debugging, branch merging, and (2) test selection at the function-level granularity. Understanding code changes underpins many important developer scenarios, e.g., debugging, version management. Code duplication amplifies the difficulties of change understanding and strengthens the needs of understanding the clone-evolution situation across changes (by detecting code clones across a previous version and the current version of the code base). Here are two example scenarios of understanding code changes in the context of code clone/duplication. First, a piece of code is duplicated/copied to another place in a change; in addition to knowing that there are added/modified lines of code in a source file, it is important to know that it is a duplication from another file/function. Second, a piece of code is changed for bug fixing where there are multiple variances in the code base; in addition to knowing that there is a change, it is important to know whether all copies of the same piece of code are changed, to prevent incomplete bug fixing.

In change understanding, we tackled a problem of efficiently and intuitively visualizing the code changes after the analysis result becomes available. We designed a tool to show the code changes and their characterization, in a classic tree-list view. The left pane shows the overall change information and the right pane is a side-by-side view. In the left pane, the code changes are organized by logical hierarchy (namespace/class/-function) in function granularity. The change type and the significance are labeled and highlighted so that the change reviewers could get a quick preview and summary before rushing into the code details.

The code difference in the evolution of a function may be of the most interest to developers. To help reviewers catch the key point of the change, we designed a token-based, side-by-side view to intuitively highlight the code changes between two versions of the function. This side-by-side mode is sometimes more efficient and clearer than the traditional inline mode especially when the differences are out of alignment in file-level comparison and the change is quite mixed and complicated.

In general, two clone functions in the same version of source files are duplication whereas a clone between two versions of source files could have resulted from a change on the same function across the two versions. Based on this insight, we could distinguish code duplication from code-version changes. our change-analysis engine (based on XIAO) goes through three steps. First, our engine indexes the code changes with a lightweight parser (which can analyze partial code). Second, our engine tries to match the indexed functions based on the function signatures and clone-detection result to get a mapping between the previous and current versions. Since we can get the similarity metric for a function's two versions, we could know whether the function has been changed and how much it is changed and further more, whether it is moved. Third, based on the mapping from the previous step, our engine categorizes the functions by a decision tree into NOT change, edited, deleted, new, moved, duplicated or renamed.

VI. Related Work

There are various experiences reported on successful technology transfer of software-engineering tools [4], [11], [14]–[20]. In this section, we discuss those previously reported experiences most related to our experiences reported in this paper.

Some of our XIAO experiences overlap with the experiences on technology transfer of Pex [19], an automated test-generation tool based on dynamic symbolic execution, in that both the XIAO project and the Pex project were conducted in a research division of a company, and both projects transferred a research tool to be shipped as part of

an industrial IDE (Pex was released as the IntelliTest feature of Visual Studio 2015). However, besides being shipped with Visual Studio, XIAO was also adopted in daily work of the Microsoft Security Response Center. Their Pex experiences include dealing with the "Chicken and Egg" problem (finding early tool adopters inside the company), corresponding to our experiences on interacting with product teams. Their Pex experiences include considering human factors, corresponding to our lesson learned on supporting human interactions with tool configurations and results described in Section V-B. Their Pex experiences include listening to practitioners, corresponding to our lesson learned on focusing on problems well recognized in practice described in Section V-A. Their Pex experiences additionally include other specific lessons learned such as evolving "dreams" (their target tools), performing well on best-case scenarios while averagely on worst-case scenarios (for the path-exploration heuristics in the Pex tool), dealing with tool users' stereotypical mindset or habits (in terms of user expectations on what the tool can do for the users), and collaboration with academia.

Main members of the Automating Test Automation (ATA) project [21] at IBM Research India shared five lessons learned in their technology transfer of their web-application testing tool: (1) relevance (corresponding to our focusing on problems well recognized in practice described in Section V-A), (2) cost-benefit trade-offs, (3) supporting early users (related to our experiences on interacting with product teams), (4) organizational dynamics (corresponding to our multiple lessons learned on interactions with product teams), (5) "Last mile" (related to our experiences on engineering efforts to make the tool efficient, robust, etc.).

VII. Conclusion

In this paper, we have reported the target task scenarios, tool overview, industry impacts, and lessons learned from Microsoft Research's XIAO project. XIAO has been shipped as part of Microsoft Visual Studio and has been deployed and intensively used in daily work at the Microsoft Security Response Center. According to our experiences, technology transfer is a rather complicated journey that needs significant efforts from both the technical aspect and social aspect. We hope that our reported experiences can inspire more high-impact technology-transfer research from our research community.

Acknowledgment

We thank our colleagues at Microsoft for their feedback and discussion on the XIAO project, especially Jonus Blunck, Andrew Fomichev, Shi Han, Xiaohui Hou, Peter Nobel, Landy Wang, Jinsong Yu, and Qi Zhang. We thank our (former) colleagues and interns for their contribution on the implementation of XIAO: Sanhong Chen, Yan Duan, Tiantian Guo, Shi Han, Ray Huang, Qi Jiang, Feng Li, Xiujun Li, Jianli Lin, Huiye Sun, Jinbiao Xu, Jiacheng Yao, and Chiqing Zhang. We also thank Yingjun Qiu for his contributions in conducting the XIAO project and Simone Livieri for his help on evaluations of XIAO. We thank Gong Cheng, Weipeng Liu, Gang Chen, Sadi Khan, Ian Bavey, Peter Provost, Vu Tran, Cameron Skinner, and many other engineers at Microsoft for helping the integration of the code-clone feature into Visual Studio. We thank Xu Guo, Xianjun Huang, and Andrew Bragdon for helping the post-release efforts on more transfer opportunities.

References

[1] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, Jul. 2002.

[2] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: Finding copy-paste and related bugs in large-scale software code," *IEEE Trans. Software Eng.*, vol. 32, no. 3, pp. 176–192, 2006.

[3] M. Gabel, J. Yang, Y. Yu, M. Goldszmidt, and Z. Su, "Scalable and systematic detection of buggy inconsistencies in source code," in *Proc. OOPSLA*, 2010, pp. 175–190.

[4] Y. Zhou, "Connecting technology with real-world problems - from copy-paste detection to detecting known bugs (keynote abstract)," in *Proc. MSR*, 2011, pp. 2–2.

[5] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, no. 7, pp. 470–495, May 2009.

[6] L. J. Osterweil, C. Ghezzi, J. Kramer, and A. L. Wolf, "Determining the impact of software engineering research on practice," *IEEE Computer*, vol. 41, no. 3, pp. 39–49, 2008.

[7] L. C. Briand, "Embracing the engineering side of software engineering," *IEEE Software*, vol. 29, no. 4, p. 96, 2012.

[8] D. Zhang and T. Xie, "Pathways to technology transfer and adoption: Achievements and challenges (mini-tutorial)," in *Proc. ICSE, SEIP, Mini-Tutorial*, 2013, pp. 951–952.

[9] Y. Dang, D. Zhang, S. Ge, C. Chu, Y. Qiu, and T. Xie, "XIAO: Tuning code clones at hands of engineers in practice," in *Proc. ACSAC*, 2012, pp. 369–378.

[10] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: scalable and accurate tree-based detection of code clones," in *Proc. ICSE*, 2007, pp. 96–105.

[11] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. R. Engler, "A few billion lines of code later: using static analysis to find bugs in the real world," *Commun. ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[12] D. Zhang, Y. Dang, J.-G. Lou, S. Han, H. Zhang, and T. Xie, "Software analytics as a learning case in practice: Approaches and experiences," in *Proc. MALETS*, 2011, pp. 55–58.

[13] S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie, "Performance debugging in the large via mining millions of stack traces," in *Proc. ICSE*, 2012, pp. 145–155.

[14] T. Ball, V. Levin, and S. K. Rajamani, "A decade of software model checking with SLAM," *Commun. ACM*, vol. 54, no. 7, pp. 68–76, Jul. 2011.

[15] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, and A. Teterev, "CRANE: Failure prediction, change analysis and test prioritization in practice - experiences from Windows," in *Proc. ICST*, 2011, pp. 357–366.

[16] E. Bounimova, P. Godefroid, and D. Molnar, "Billions and billions of constraints: Whitebox fuzz testing in production," in *Proc. ICSE*, 2013, pp. 122–131.

[17] D. Zhang, S. Han, Y. Dang, J. Lou, H. Zhang, and T. Xie, "Software analytics in practice," *IEEE Software*, vol. 30, no. 5, pp. 30–37, 2013.

[18] J. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Software analytics for incident management of online services: An experience report," in *Proc. ASE*, 2013, pp. 475–485.

[19] N. Tillmann, J. de Halleux, and T. Xie, "Transferring an automated test generation tool to practice: from Pex to Fakes and Code Digger," in *Proc. ASE*, 2014, pp. 385–396.

[20] C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a program analysis ecosystem," in *Proc. ICSE*, 2015, pp. 598–608.

[21] S. Chandra, S. Thummalapenta, and S. Sinha, "Lessons from the tech transfer trenches," *Commun. ACM*, vol. 59, no. 2, pp. 37–39, Jan. 2016.