Using sketch recognition for capturing developer's mental models

Tatiana De-Wyse, Emmanuel Renaux, José Mennesson

Using sketch recognition for capturing developer's mental models

Tatiana De-Wyse, Emmanuel Renaux, José Mennesson
IMT Lille Douai/CRIStAL (UMR CNRS 9189)
University of Lille, France
firstname.lastname@imt-lille-douai.fr

***Abstract*—The purpose of agile practices is to optimize engineering processes. Beside it, software documentation often suffers from the priority given to fast and successive deliveries of new functionalities. As a consequence, incomplete documentation and graphical representation make it difficult for a developer to maintain and evolve. Sketching is an integral part of the software design and development. Indeed, among other stakeholders, developers use sketches to informally share knowledge about source code. Since sketches are often hand-drawn, written on paper or whiteboard without any additional technology tool, they are not considered as an artifact in agile method or traditional engineering process but as a disposable production.**

In this work, we focus on sketches containing Unified Modeling Language (UML) diagrams. To produce documentation or to exploit information from this kind of sketches, developers have to transcribe it in a UML case tool, what they see as a waste of time that hinders productivity. We argue that sketches might be worth considering as non-code artifacts. The fact is that developer or designer drop informally a lot of information about the software, which is unusable.

Our goal is to verify that simply capturing sketches can provide useful information for later use and then improve the modeling process efficiency. In this paper, we present a preliminary approach that consists in automatically capture information from the sketches using image processing and pattern recognition. We propose a fledgling prototype that demonstrates the proposal's viability. Then, as a future work, we plan to put in the hands of the developers a finalized version of our prototype and study the added value of our proposal.

***Index Terms*—Sketch, UML, Image recognition, Engineering Process, empirical study**

I. Introduction

More and more developers are adopting agile practices, no matter if it is a fad or a pragmatic evolution of methodologies. In agile approaches, the source code is considered as the most valuable artifact. It leaves aside other means to describe the architecture or functionalities like models and other human-readable documents. Nevertheless, developers create and maintain non-code artifacts [14] when developing and evolving software. These non-code artifacts help to express the requirements, understand the software architecture or document the source code. Each one is dedicated to specific stakeholders and allows to communicate, exchange and maintain the system. The Unified Process [22] precisely defined different types of artifacts throughout the project lifecycle, in order to ensure high-quality software production. The agile methodScrum [33], implies more roughly documents like user stories, programs, tests cases, and other deliverables created before designing and implementing the system.

Among these artifacts, graphical representations are widely used to model the system, as in Industrial Software Development [40]. UML[1] [31] is a standard modeling language and a graphical representation that fosters communication and visualization of software architecture. As a graphical representation of the system, it helps to understand, discuss or brainstorm about it. The strength of UML is that any developer understands it since he learned it at school and it is the de facto standard for modeling and designing information system.

However, UML is often subject to negative press and is considered as a waste of time. But surveys like [6], [29] highlight that although many engineers say they do not use UML, they use it informally. In fact, engineers actually admit doing sketches of UML diagrams [2], [28] during activities like:

- Sharing knowledge about the code in collocated design collaborations [10]
- Exchanging ideas in design software session with easy transitions between different types of sketches like lists, tables, GUIs, ER diagrams, class diagrams, code, drawings. Sketches support mental simulations, reviews of designers progress and discussions of alternative [28]
- Creating informal graphical representations to quickly document workflows. Sketches help the comprehension of code [2]
- Supporting their development workflows by helping them for different purpose and context [43]
- Finally, externalizing their mental models of source code. Tools do not allow to support face-to-face communication [9]

In the next section, we highlight some obstacles to the adoption of UML in traditional engineering practices. Since there is little documentation of the software architecture, in section III, we justify the use of sketches to capture developers' mental models of their source code. Related work focuses on sketching tools in section IV. In section V, we introduce our prototype that captures and understand sketches without any effort from the developers. In the prototype, we firstly restricted our focus to the UML class diagram. Finally, we discuss a survey we need to further validate our work and the

[1]Unified Modeling Language

next steps that have to be created to prove that capitalizing on sketches can improve the development process efficiency.

II. Weak adoption of UML in IT projects

Documentation and modeling tasks require significant time to be produced and maintained. IT project engineers do not maintain quality models since they see them as a waste of time and effort. Although this proposal is controversial [29], [40], developers prefer to use this time and effort in implementing new functionalities, passing tests and solving problems. Moreover, agile practices do not help the adoption of UML as they speed up the pace of deliveries. To counter these adverse developments, we selected in the literature, three obstacles to the adoption of UML or other graphical representations in IT projects. Sketching is the more natural way to draw UML models, even if they are not as formal as in a UML tool. We think and want to verify that more sketches in projects and a simply manner to exploit them can help to counter these obstacles, and reintroduce software architecture modeling as an efficient practice.

A. Operational software rather than comprehensive documentation

Agile practices encourage short development cycles to produce a shippable product increment. In these methods highly used nowadays, documentation and models are seen as a burden artifact [38]. Hence, the code and testing have taken the leadership over exhaustive documentation and graphical design of the software architecture. In terms of tools, teams mainly use development tools and project management tools like Jira[2]. In a simplistic view, the only non-code artifacts are the user stories. They make it possible to organize sprints and collaboratively monitor the project on a Kanban board. Globally, agile processes leave little room for modeling tools and models. Quick brainstorming sketches contain information about the source code architecture. It would be nice to be able to easily retrieve some of this information.

B. Inadequacy of Point&Click tools

Even if project management tools are used, in [1] Azizyan et al. show that collocated teams prefer physical walls and paper. Developers also not use Point and Click solutions to model or maintain software architecture and its behavior but prefer to use paper or whiteboard. They find them more flexible [34] as paper and whiteboard allow developers to use their own graphical representation (informal notation). They used the sketch as memory help for debugging, explaining task and refine or improve their idea [43]. Then, if it is required it allows the preparation of formal documentation and developers transcribe sketches in a formal way [2]. Sometimes, to deal with the lack of documentation of the source code, software visualization tools and reverse engineering tools automatically provide the UML representation of code for better understanding and localization of code source. For example, this helps developers to take in hand unfamiliar code [25]. Finally, Point & Click diagramming tool suffers from the lack of collaborative capacities. In collaborative activities, co-workers need awareness about artifacts and other coworkers. They need information about who, how and why things have been done and then they have to exchange with other co-workers [21]. The use of these point & click solutions could be reduced by trying to retrieve both the information contained in the sketches and other contextual information.

[2]https://fr.atlassian.com/software/jira

C. Cognitive shift due to the proliferation of tools

The need for knowledge exchange and code explanation fosters the use of communication and social tools [39]. This increases even more, the number of tools used by developers. Switching from one tool to another or interruption due to instant messaging, for instance, cause cognitive shift and knowledge loss. This reduces the efficiency of the development process and increases the feeling of a waste of time. Surveys like [14] show that existing tools do not provide sufficient support for carrying out some tasks like visualizing artifacts that are larger than their screen and with interconnected artifacts that have to be accessed simultaneously. The methods to deal with that problem are inefficient and frequently error-prone. By reducing the number of tools and exploiting the engineer's workstation equipped with a keyboard and draft paper containing his sketches, we wish to limit the cognitive shift.

Let's see in the following how we plan to counter these obstacles through a better consideration of sketches.

III. Capturing mental model through sketches recognition

In traditional development processes, lack of documentation is a factor of loss in efficiency. For the reasons we gave previously, developers do not produce an exhaustive UML model of their source code. However, they create and maintain mental models of the architecture of the code. Inevitably there is a loss of code knowledge and developers spend time understanding and trying to recover the rationale behind the code. They rely on and interrupt their co-workers to find answer when they get stuck by a task [24]. The consequence is that the code becomes hard to debug and complex to maintain [12].

Understanding the rationale behind source code is the biggest problem for developers. Graphical representations help to understand, to discuss, to brainstorm, etc. Sketches are an example of informal visualizations that are often created when understanding or explaining the source code [2]. Moreover, hand-drawn is a natural and flexible way to model a domain. And even if the sketch do not respect formal notation of modeling language they foster later their creation. Contrary to the point & click solution that requires the respect of their notation, hand-drawn do not constraint developers. Surveys have revealed that developers make sketches, which often contain UML diagrams [2], [40].

We base our work on the fact that sketches and diagrams give an accurate picture of parts of the developers' mental model and help to understand a software project [24]. But since the sketch life cycle is short [43], our goal is to design a tool that could help us to analyze transparently the sketches and to exploit the information contained. We want to improve the software development process using graphical representation more efficiently with a minimum of effort and without disturbing the development activity. The key success factor is for us: capturing the mental model of the developers in their sketches through sketch recognition.

IV. Related Work

There are a lot of sketching tools since the different devices allowing sketching activity are numerous. In software design and development practices, only a few of them allow the recognition of sketch elements and their semantic meaning as shown in table I. Here is a list of available sketching tools we found in a comprehensive survey of the literature.

A. Tools allowing digital sketching activity

Digital Sketching defines a sketching activity realized thanks to an electronic device like digital pen or finger on a touch screen, or mouse for computer desktop. This category also contains e-whiteboard[3] with the hand gesture or digital pen.

a) Formal early design: This category of tools allows expert designers equipped with an electronic device to freely create sketches that will be transformed automatically in formal models.

SUMLOW [8] is a Visual Basic application for early design stage sketching of UML diagrams on e-whiteboard. It uses Rubine Algorithm for text recognition and multi-gesture algorithm for shape recognition. It allows annotation of the digital sketch drawn and their format transformation to be used later in a case tool.

MaramaSketch [16] is an extension of a Marama meta-tool, a set of core Eclipse[4] plug-ins that provides diagram and model management. It allows sketch drawn with tablet PC stylus, mouse, e-whiteboard pen. Recognition of shape and text element on a sketch is made by HHREco toolkit[5].

Tahuti [17] allows a user to draw UML class diagram with a computer mouse. Recognition of geometrical element of a class diagram is performed by analyzed stroke made by the user when he is drawing on Tahuti.

In [23], Lank et al. present a prototype that allows to drawn sketches thanks to e-whiteboard, tablet, and mouse. Recognition of what the user has drawn is made by analyzing pen stroke. Segmentation and correction of what is drawing are realized by the user to let him drawn freely without any computation time feeling.

The drawbacks of these tools are: first, developers need to be equipped with an electronic device, and second, the original sketches are lost, and also a lot of information expressed informally.

b) Informal early design: Informal sketching tool fosters the communication and collaboration between different stakeholders.

Calico [27] is a system for e-whiteboard which proposes a canvas for managing and drawing a digital sketch. No recognition of what is drawn is provided and links between sketches have to be done manually by the user.

Gambit [32] supports collaborative early design session of user interface (UI). Designed according to a client-server model, it can store and retrieve digital hand-drawn sketches. Users of Gambit can annotate and use it on a tablet, desktop, and video-projector.

AugIr [20] provides an interaction room system by combining multiple e-whiteboards. Stakeholders can manipulate hierarchical canvas and freely organize their views. Size, direction and temporal context of drawn stroke are analyzed to recognize meaningful text and match the artifact with link automatically.

IdeaVis [13] is a tool for co-located sketching sessions. It uses digital paper and pen for sketch creation and manipulation on large interactive display for presentation.

FlexiSketch [44] and his extension FlexiSketch Team [45], allows collaborative sketching. Symbols recognized are manually pre-defined by user and store in a library for later re-use and recognition if it is redrawn by users.

On the one hand, informal sketching fosters the communication and collaboration of stakeholders by providing a totally informal way of explaining ideas and brainstorm. On the other hand, this type of tool does not take into account the hidden figures, the developers, the code and the analog sketches done beneath the design session.

c) Code explanation/modeling: Tools built on code explanation and modeling increase the comprehension of the rationale behind the code written by developers.

CodeGraffiti [26] is a prototype for Adobe Bracket[6] which links informal digital sketch on a tablet and smartphone to source code artifact.

OctoUML [42] supports collaborative UML modeling during any phase of the software design process, in allowing formal and informal notations on a touchscreen, e-whiteboard, and computer with mouse interaction. It allows connection with version control and source code management.

Even if these tools step the code explanation and documentation up, analog sketches are totally dismissed. Furthermore, developers that doing sketches, need to take in hand a new tool and redraw every diagram created on analog support.

B. Tools allowing Analog Sketching

Analog sketching tools allow taking into account the whole sketch.

a) Informal code documentation: SketchLink [4] is based on a server, a web application and an IDE (Integrated Development Environment) plugin [7] to allow the sketching

[3]electronic whiteboard

[4]http://www.eclipse.org/

[5]https://ptolemy.berkeley.edu/projects/embedded/research/hhreco/

[6]http://brackets.io/

[7]https://www.jetbrains.com/idea/

		SUMLOW	LivelySketch	MaramaSketch	GAMBIT	IdeaVis	FlexiSketch	OctoUML	Calico	CodeGraffiti	SketchLink	AugIr	Tahuti	Lank's POC
Analog sketch	Paper		✔			✔					✔			
”	Whiteboard										✔			
Digital sketch	E-Whiteboard	✔		✔				✔	✔			✔		✔
”	Touch Screen							✔	✔					
”	Tablet		✔	✔	✔	✔	✔		✔	✔	✔			✔
”	Smartphone				✔					✔	✔			
”	PC	✔	✔	✔	✔		✔	✔		✔	✔		✔	✔
”	Stylus	✔		✔	✔	✔				✔				
Online sketch recognition		✔		✔	✔	✔	✔				✔		✔	✔

TABLE I: Table of tools supporting sketching activity

activity on source code artifact. Analog and Digital sketch can be stored and have to be manually linked on Java code source project with the rectangular marker. It fosters HTML documentation creation on the source code.
LivelySketch [3] enables the capture and the annotation of analog sketch and diagram by provided a REST API[8]. Identification of a sketch is done by a printed QR code with predefined UUID (Universally Unique Identifier). Links between sketches need to be done manually by touch gesture and/or mouse.
Although the sketches are captured and stored, there is another step required to fully take into account the meaning of sketches drawn.

All these tools are the example of the wide variety of devices and technologies that can foster the documentation in any step of the software development process. However, if the workplace is not fit with them, sketching with these tools is useless [36].

The majority of the tools in this related work use online recognition to retrieve information present in a sketch. Indeed, online recognition is done by real-time analysis of the stroke made by the user when he draws on an electronic device. Offline recognition is another challenge. Unlike the online approach, offline recognition needs to capture and digitize the sketch before being analyzed. This analysis consists of a combination of many image processing algorithms. This complex mining activity results in finding elements contained by the image. Some methods dealing with digitized documents recognition in other domain like architectural plan [15], electronic diagram [30], flowcharts [18], and geometric shape [37] can be found. This reveals the possibility of recognizing UML elements on a bitmap image.

V. Our approach

In [7] Chaudron and Jolak revealed their vision of a new generation of software design environments. We try to approach this vision by raising analog sketches to the level of project artifacts. We choose offline mode 1) to let the designer uses any kind of support and 2) to not impose him yet another tool to manage. To this end, analog sketches need to be digitized with for instance a smartphone and recognized to understand part of their semantic meaning. To partly respond to the need for collaborative capacities like awareness about artifacts and other coworkers, some metadata can be added to the sketch automatically or manually to better identify its context.

Our goal is to use sketches to capture efficiently the mental model of the developer without asking him to transcribe it. Analyzing digitized hand-drawn sketches is a complex task [5]. Contrary to digital sketching, sketches on paper and whiteboard are considered as offline documents. In this sense, we do not have any information on the real-time tracking of the pen when shapes or text are drawn. So, it is difficult to segment the image and to recognize shapes and text which are not normalized. Finally, multiple challenges exist with a hand-drawn document:

- handwriting style differs from one person to the next,
- non-uniformity in size, shape, and orientation of hand-written elements,
- text and shape separation due to touching elements,
- text typography does not respect any known font.

In this section, we propose an empirical approach based on classical tools in image processing to conduct a proof of concept. The first step is the pre-processing. The image needs to be cleaned to have the correct format to optimize image processing algorithm.

A. Pre-Processing phase

Fig. 1: Prototype advancement

In our study, a simple example of a class diagram (Fig. 2) is used to understand what kind of processing we need to employ. Indeed, components of hand-drawn class diagram depend on the way the user has drawn it. For example, boxes representing classes and lines representing links between classes can vary a lot from a user to another. In this context, extracting straight lines using classical methods as Hough transform [11] are no longer efficient as lines are more or less straight.

[8]Application programming interface Representational State Transfer

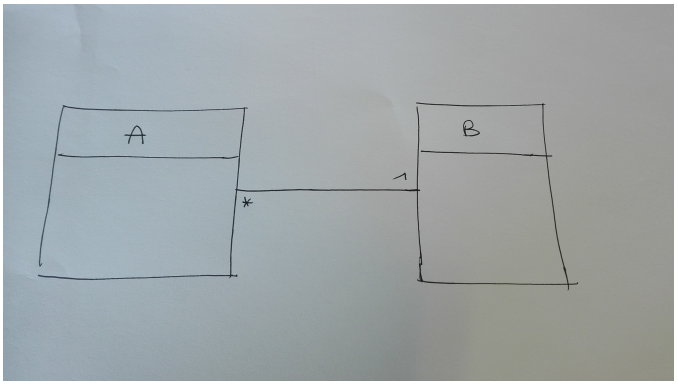

Fig. 2: Picture of a simple class diagram

To do that, OpenCV [9] and Python [10] are used to prepare our image for extraction of text and shapes drawn in our class diagram. Fig. 1 represents the pre-processing steps which consist in a binarization of the image (Fig. 3 (a)) followed by a segmentation of shapes and text.

B. Shapes and text separation

In the case of hand-drawn elements, like in Fig. 2, contours of class boxes are not closed. Morphological closing [35] is used to obtain connected components (Fig. 3(b)). Suzuki algorithm [41] is then used to retrieve contours from the image (Fig. 3 (c)).

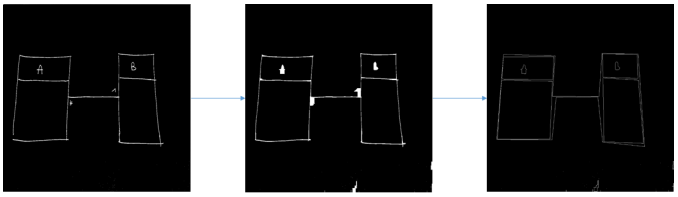

Fig. 3: (a) binary image, (b) connected components extracted, (c) contours extracted

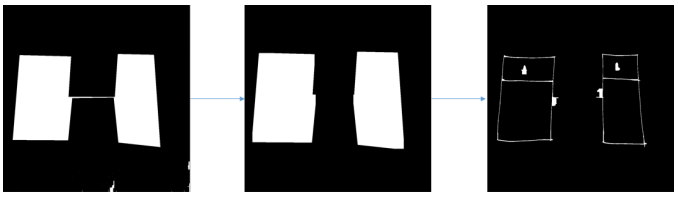

Fig. 4: (a) filled shapes (b) morphological opening (c) classes extracted

[9]https://opencv.org/
[10]https://www.python.org/

As there is a line between classes, only one connected component is obtained for the two classes and the association between them (Fig. 4(a)). In order to separate them, connected components are filled (Fig. 4(b)) and a morphological opening is performed. By using the previous image as a binary mask on image obtain in 3(b), only classes are obtained (Fig. 4(c)).

Fig. 5: Segmentation result of class diagram

C. Segmentation

Finally, thanks to all step realized before, we are able to segment a class diagram into elements representing individual class box (Fig. 5(a)), class box component containing the class name (Fig. 5c) and class box component containing attributes name (Fig. 5b). Classical recognition methods can be then used to recognize shapes and classical Optical character recognition (OCR) as Tesseract [11] can be used to recognize the hand-written class name (Fig. 5).

VI. Future Works

A. Finalization of the sketches recognition tool

The purpose of the work presented in this section was to elaborate the specifications of our sketch recognition mechanism. For the moment, we have only made a very simple prototype to validate that it is possible to automatically extract information from the sketch. The few tests carried out have been promising. So we plan the future steps to continue developing our tool as shown in Fig 6. First, we will need to complete the segmentation of shapes and text in more complex diagrams. Then Tesseract engine will be used to perform OCR recognition of names, methods, attributes, cardinalities. For shape features recognition, we will need to build a corpus of hand-drawn class diagrams. Ho-Quang et al. [19] have built a similar base of digitized images of class diagrams built with UML tools and found on the internet. Our base of hand-drawn CD diagrams will be used to train a classification algorithm analyzing shape elements to extract the information concerning size, orientation and type. Then, this trained classifier will be able to identify elements of a CD diagram. Once completing these steps and combining their results, we hope to obtain sketches with augmented information.

[11]https://github.com/tesseract-ocr/tesseract

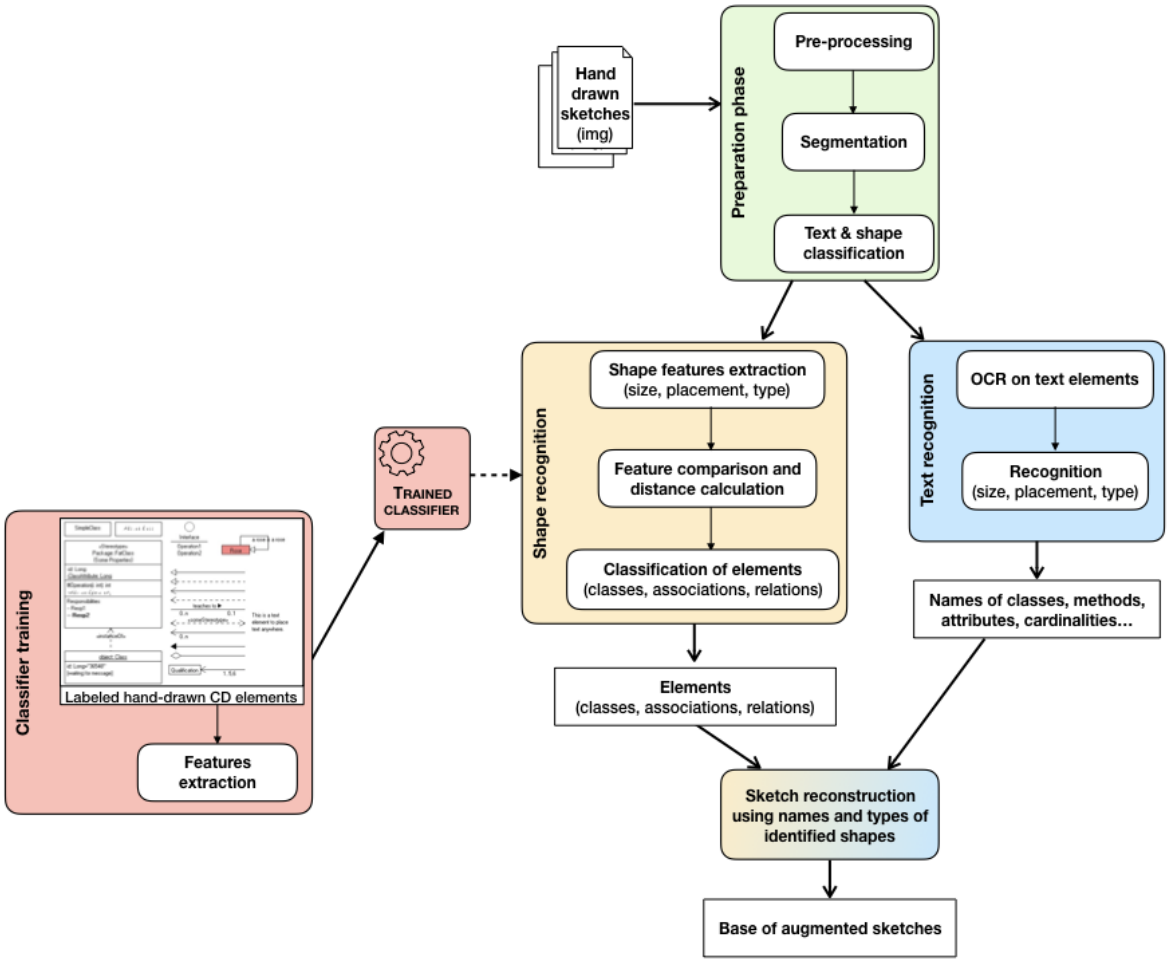

Fig. 6: Overview of the entire future prototype

B. Experimentation

The goal of our work is to verify that more sketches in projects and our recognition and classification mechanism can help to reintroduce software architecture modeling as an effective practice in the engineering project.

We plan to do an experimentation and evaluation of our prototype with IT professionals and students. The first step is to evaluate the interest of an assisted archiving of the sketches. The second one is to evaluate the added value of later retrieval in a new project. And the third one is to evaluate the added value of the archived sketch to understand a code that the engineer did not write, for instance, a new team member.

a) Assisted archiving: We are designing our tool with ease of use in mind. We will ask the practicing software engineer to photograph his sketch with its smartphone. Our tool will archive the sketch recognizing the maximum of content semantic (shape, text, etc.) and adding context data from other tools like the code editor, the project calendar, etc. The first part of the evaluation is to observe the developer using our prototype, to find out if it is easy to use and if it helps to increase the use of sketches in IT projects, and if it encourages improvement of the quality of the sketches. And finally, we want to know if the engineer accepts to use and disseminate his sketches as documentation of its code.

b) The case of a new project: In the case of a practicing engineer brainstorming on a new project, we experiment with two types of engineer, the first one starts to draw sketches as he used to do, the other receives help from our tool. In the second case, he takes a photo of the draft of a sketch and the tool shows him all the corresponding sketches. Will it be a help for an engineer? And what use does the engineer have of it?

c) The case of a new team member: In the same way, we will compare two types of engineer. The first one tries to discover the code without help. The other selects a piece of code and our tool shows him the corresponding sketches.

VII. Conclusion

In this paper, we list some known reasons why documentation and graphical representation are important for the

software to be designed and maintained. We focused on UML models. Then we select in the literature some obstacles to the adoption of UML diagrams to accompany the software development process. Our motivation is to verify the idea that using hand-drawn sketches that every developer naturally does, can improve traditional development processes or agile approaches. To set up experiments we elaborate a mechanism simply to use without adding any cognitive load to the developers. This mechanism uses off-line sketches recognition techniques. We describe our simple prototype that confirms we can automatically extract information from the sketch. So we plan the future steps to continue developing our tool. Finally, we present as a future work different cases we want to experiment to check the viability of our proposition in actual IT projects and to verify if our proposition really improves the quality and effectiveness of software modeling process.

REFERENCES

[1] Gayane Azizyan, Miganoush Katrin Magarian, and Mira Kajko-Mattsson. Survey of agile tool usage and needs. In *2011 Agile Conference, AGILE 2011, Salt Lake City, Utah, USA, August 7-13, 2011*, pages 29–38, 2011.

[2] Sebastian Baltes and Stephan Diehl. Sketches and Diagrams in Practice. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 530–541, 2014.

[3] Sebastian Baltes, Fabrice Hollerich, and Stephan Diehl. Round-trip sketches: Supporting the lifecycle of software development sketches from analog to digital and back. *CoRR*, abs/1708.01787, 2017.

[4] Sebastian Baltes, Peter Schmitz, and Stephan Diehl. Linking sketches and diagrams to source code artifacts. *CoRR*, abs/1706.09700, 2017.

[5] Showmik Bhowmik, Ram Sarkar, Mita Nasipuri, and David S. Doermann. Text and non-text separation in offline document images: a survey. *IJDAR*, 21(1-2):1–20, 2018.

[6] Michel R. V. Chaudron, Werner Heijstek, and Ariadi Nugroho. How effective is UML modeling ? - an empirical perspective on costs and benefits. *Software and System Modeling*, 11(4):571–580, 2012.

[7] Michel R. V. Chaudron and Rodi Jolak. A vision on a new generation of software design environments. In *Proceedings of the First International Workshop on Human Factors in Modeling co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), Ottawa, Canada, September 28, 2015.*, pages 11–16, 2015.

[8] Qi Chen, John C. Grundy, and John G. Hosking. SUMLOW: early design-stage sketching of UML diagrams on an e-whiteboard. *Softw., Pract. Exper.*, 38(9):961–994, 2008.

[9] Mauro Cherubini, Gina Venolia, Robert DeLine, and Andrew J. Ko. Let's go to the whiteboard: how and why software developers use drawings. In *Proceedings of the 2007 Conference on Human Factors in Computing Systems, CHI 2007, San Jose, California, USA, April 28 - May 3, 2007*, pages 557–566, 2007.

[10] Uri Dekel and James D. Herbsleb. Notation and representation in collaborative object-oriented design: an observational study. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada*, pages 261–280, 2007.

[11] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, 1972.

[12] Ana M. Fernández-Sáez, Danilo Caivano, Marcela Genero, and Michel R. V. Chaudron. On the use of UML documentation in software maintenance: Results from a survey in industry. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, pages 292–301, 2015.

[13] Florian Geyer, Jochen Budzinski, and Harald Reiterer. Ideavis: a hybrid workspace and interactive visualization for paper-based collaborative sketching sessions. In *Nordic Conference on Human-Computer Interaction, NordiCHI '12, Copenhagen, Denmark, October 14-17, 2012*, pages 331–340, 2012.

[14] Parisa Ghazi and Martin Glinz. Challenges of working with artifacts in requirements engineering and software engineering. *Requirements Engineering*, 22(3):359–385, Sep 2017.

[15] Achraf Ghorbel, Aurélie Lemaitre, Éric Anquetil, Sylvain Fleury, and Eric Jamet. Interactive interpretation of structured documents: Application to the recognition of handwritten architectural plans. *Pattern Recognition*, 48(8):2446–2458, 2015.

[16] J. Grundy and J. Hosking. Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. In *29th International Conference on Software Engineering (ICSE'07)*, pages 282–291, May 2007.

[17] Tracy Hammond and Randall Davis. Tahuti: A geometrical sketch recognition system for uml class diagrams. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[18] Jorge-Ivan Herrera-Camara and Tracy Hammond. Flow2code: From hand-drawn flowcharts to code execution. In *Proceedings of the Symposium on Sketch-Based Interfaces and Modeling*, SBIM '17, pages 3:1–3:13, New York, NY, USA, 2017. ACM.

[19] T. Ho-Quang, M. R. V. Chaudron, I. Samelsson, J. Hjaltason, B. Karasneh, and H. Osman. Automatic classification of uml class diagrams from images. In *2014 21st Asia-Pacific Software Engineering Conference*, volume 1, pages 399–406, Dec 2014.

[20] Markus Kleffmann, Sebastian Rohl, Matthias Book, and Volker Gruhn. Evaluation of a traceability approach for informal freehand sketches. *Autom. Softw. Eng.*, 25(1):1–43, 2018.

[21] Andrew J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007*, pages 344–353, 2007.

[22] Philippe Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000.

[23] E. Lank, J. Thorley, S. Chen, and D. Blostein. On-line recognition of uml diagrams. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 356–360, 2001.

[24] D. Latoza, Thomas, Gina Venolia, and Robert Deline. Maintaining mental models: a study of developer work habits. *Proceedings of the 28th International Conference on Software Engineering*, pages 492–501, 2006.

[25] Seonah Lee and Sungwon Kang. What situational information would help developers when using a graphical code recommender? *Journal of Systems and Software*, 117:199–217, 2016.

[26] Leonhard Lichtschlag, Lukas Spychalski, and Jan O. Borchers. Codegraffiti: Using hand-drawn sketches connected to code bases in navigation tasks. In *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2014, Melbourne, VIC, Australia, July 28 - August 1, 2014*, pages 65–68, 2014.

[27] Nicolas Mangano, Thomas D. LaToza, Marian Petre, and André van der Hoek. Supporting informal design with interactive whiteboards. In *CHI Conference on Human Factors in Computing Systems, CHI'14, Toronto, ON, Canada - April 26 - May 01, 2014*, pages 331–340, 2014.

[28] Nicolas Mangano, Thomas D. LaToza, Marian Petre, and André van der Hoek. How software designers interact with sketches at the whiteboard. *IEEE Trans. Software Eng.*, 41(2):135–156, 2015.

[29] Marian Petre. UML in practice. In *Proceedings - International Conference on Software Engineering*, pages 722–731, 2013.

[30] Mahdi Rabbani, Reza Khoshkangini, H. S. Nagendraswamy, and Mauro Conti. Hand drawn optical circuit recognition. In *Proceeding of the Seventh International Conference on Intelligent Human Computer Interaction, IHCI 2015, Allahabad, India, 14-16 December 2015.*, pages 41–48, 2015.

[31] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004.

[32] Ugo Braga Sangiorgi, François Beuvens, and Jean Vanderdonckt. User interface design by collaborative sketching. In *Designing Interactive Systems Conference 2012, DIS '12, Newcastle Upon Tyne, United Kingdom, June 11-15, 2012*, pages 378–387, 2012.

[33] Ken Schwaber. Scrum development process. In Jeff Sutherland, Cory Casanave, Joaquin Miller, Philip Patel, and Glenn Hollowell, editors, *Business Object Design and Implementation*, pages 117–134, London, 1997. Springer London.

[34] T. Sedano, P. Ralph, and C. Péraire. Software development waste. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 130–140, May 2017.

[35] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983.

[36] David Socha and Josh D. Tenenberg. Sketching and conceptions of software design. In *8th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2015, Florence, Italy, May 18, 2015*, pages 57–63, 2015.

[37] Dan Song, Dongming Wang, and Xiaoyu Chen. Retrieving geometric information from images: the case of hand-drawn diagrams. *Data Min. Knowl. Discov.*, 31(4):934–971, 2017.

[38] Christoph Johann Stettina and Werner Heijstek. Necessary and neglected?: An empirical study of internal documentation in agile software development teams. In *Proceedings of the 29th ACM International Conference on Design of Communication*, SIGDOC '11, pages 159–166, New York, NY, USA, 2011. ACM.

[39] Margaret-Anne D. Storey, Alexey Zagalsky, Fernando Marques Figueira Filho, Leif Singer, and Daniel M. Germán. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Trans. Software Eng.*, 43(2):185–204, 2017.

[40] Harald Störrle. How are conceptual models used in industrial software development?: A descriptive survey. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE 2017, Karlskrona, Sweden, June 15-16, 2017*, pages 160–169, 2017.

[41] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.

[42] B. Vesin, R. Jolak, and M. R. V. Chaudron. Octouml: An environment for exploratory and collaborative software design. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 7–10, May 2017.

[43] Jagoda Walny, Jonathan Haber, Marian Dörk, Jonathan Sillito, and M. Sheelagh T. Carpendale. Follow that sketch: Lifecycles of diagrams and sketches in software development. In *Proceedings of the 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2011, Williamsburg, VA, USA, September 29-30, 2011*, pages 1–8, 2011.

[44] Dustin Wüest, Norbert Seyff, and Martin Glinz. Flexisketch: A mobile sketching tool for software modeling. In *Mobile Computing, Applications, and Services - 4th International Conference, MobiCASE 2012, Seattle, WA, USA, October 11-12, 2012. Revised Selected Papers*, pages 225–244, 2012.

[45] Dustin Wüest, Norbert Seyff, and Martin Glinz. FLEXISKETCH TEAM: collaborative sketching and notation creation on the fly. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*, pages 685–688, 2015.