Poster: Incremental UML for Agile Development with PREXEL

Eric Braude
Boston University
Boston, MA
USA
ebraude@bu.edu

Jason Van Schooneveld
Boston University
Boston, MA
USA
jasonvan@bu.edu

ABSTRACT

UML creates useful visualizations but they become monolithic, complex, and expensive to maintain. In agile development, documentation is secondary, which discourages the use of UML even further. We introduce an in-code, just-in-time, maintainable approach to UML, supported by a tool called PREXEL. PREXEL minimizes interruptions in coding by allowing concise in-line specifications which automatically synthesize in-code graphical ASCII class models, class and method skeletons, and class relationships.

CCS CONCEPTS

• **Software and its engineering** → Software design engineering; Software design techniques; Object oriented development

KEYWORDS

embedded UML, agile UML, incremental UML, inline UML, agile modelling[1]

ACM Reference format:

Eric Braude, Jason Van Schooneveld. SIG Proceedings Paper in word Format. In *Proceedings of 40th International Conference on Software Engineering, Gothenburg, Sweden, May-June 2018,* 2 pages.

1 INTRODUCTION

In agile development, UML is often abandoned due to the expense of creating and maintaining it, and because there are few ways to manage UML fragments. We introduce a response: a method, and an accompanying tool called PREXEL (Programming Execution with fragmented UML), for just-in-time, just-enough UML that's embedded in the code itself (https://github.com/jasonvan/prexel). PREXEL minimizes interruptions in coding by allowing the developer to specify classes, methods, and relationships compactly, inline, and on the fly. PREXEL automatically synthesizes class models in ASCII at the points of specification, together with class skeletons, method skeletons, and class relationships.

2 LITERATURE ON EMBEDDED UML

The practical integration of UML in agile development is discussed, for example, by Wei et al [7]. Pitkänen and Selonen study incremental modeling in [6]. Marovac [5] shows similar motivations but does not show UML figures. Ambler [1] discusses agile modeling but not embedding UML as introduced here. Karagiannis [4] gives a concrete example with "Agile Modeling Method Engineering," but at a high level, unlike our approach.

3. PREXEL

Our goal is just-in-time design to support agile implementation, and our scenario is an agile developer in the midst of writing a function. When the developer actually requires classes, with inter-relationships such as inheritance, attributes, or methods, these should be introduced and visualized there and then with minimal interruption. UML parts should be diagrammed if, and only if deemed helpful by the developer and anticipated readers of the code.

As an example, suppose that in the course of programming, a developer decides that a *Kitchen* class is desired, inheriting from class *Room*, with attribute *square_feet* and methods *arrange_kitchen*(), *place_floor_cabinet*(), and *place_wall_cabinet*(). The developer can enter the following PREXEL at that point in the current code:

```
#  |Room >> Kitchen square_feet arrange_kitchen()
```

Upon hitting a control character, PREXEL performs three actions: (1) it replaces the above with the version below, within comments;

[1] Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194985

```
 ____
|Room|
|____|
△
|__________________
|     Kitchen      |
|------------------|
|square_feet       |
|arrange_kitchen()|
|__________________|
```

(2) it creates files with the specified class and method skeletons, and (3) it creates XML files expressing *Room* and *Kitchen*.

4 EXAMPLE IMPLEMENTATION OF PREXEL

We will take as example an application called REWOTS (Realtime Workday Task Scheduler), which prioritizes a list of everyday
tasks. It updates automatically at regular intervals, recalculating
priorities as deadlines get closer. We show PREXEL in action while writing function *prioritize_tasks()*. As anticipated in [3],

PREXEL facilitates this inline, as follows:

```
'''
|TaskList <>-the_tasksS---*> Task
|get_the_tasks()
|prioritize()
'''
```

The "S" suffix denotes a static member. The developer is free to specify this via smaller fragments as follows:

```
'''
|TaskList get_the_tasks() prioritize()
|TaskList <>-the_tasksS---*>Task
'''
```

PREXEL supplies one UML figure for each single comment line (starting with '#') and each grouping within triple quotes (in the case of Python). Interpreting PREXEL in the example above results in the creation of the classes *TaskList* and *Task* as specified. PREXEL also replaces the specifications above with a prettyprinted version as follow:

```
'''
 _________________                 ____
|    TaskList     |<>-the_tasksS---*>|Task|
|-----------------|                  |____|
|get_the_tasks()S|
|prioritize()S    |
|_________________|
'''
```

The developer is assured of the existence and placement of their desired functions, and can thus proceed to code with them in prototype, such as the following:

```
TaskList.get_the_tasks()
return TaskList.prioritize()
```

UML can also be prettyprinted from single-line specification, as in the example below.

```
# |TaskList <>-the_tasksS--->Task

'''

 _________                  ____
|TaskList|<>-the_tasksS--->|Task|
|_________|                |____|

'''
```

5 EFFECTIVENESS OF PREXEL

In conventional development, the user creates classes, methods, and class relationships by typing them in separate files, and typing the corresponding code. In PREXEL-enabled development, the user types PREXEL specifications for class model fragments as needed, within the code with the need. This takes less time. Table 1 compares conventional with PREXEL-enabled development and suggests the superiority of the latter.

	Conventional	PREXEL-enabled
Code for function and referenced classes	Y slower	Y faster
UML generated automatically from source code	Y (but incomplete for fragments and dynamically typed languages)	Y
Automatically prettyprinted UML parts as desired	N	Y

Table 1: Comparison of conventional vs. PREXEL-enabled development

4 CONCLUSIONS

We believe that PREXEL creates preferred visual documentation (particularly embedded fragmented UML), and takes less time.

With regard to limitations of this work note that our claims of improvement are only now in the process of being checked by experiment. Also, developers will modify code generated by PREXEL, and we have not yet fully accounted for this.

REFERENCES

[1] Ambler, S. Agile/Lean Documentation: Strategies for Agile Software Development.

http://agilemodeling.com/essays/agileDocumentation.htm#ModelsDocumentsSourceCode.

[2] E. Braude. "Incremental UML for Agile Development: Embedding UML Class Models in Source Code," RCoSE 2017 - 3rd International Workshop on Rapid Continuous Software Engineering 2017-05-22 Buenos Aires, 2017.

[3] D. Karagiannis. Agile modeling method engineering. 2015, doi: 10.1145/ 2801948.2802040.

[4] N. Marovac. "UML based embedded documentation for semi-automatic software development," SIGSOFT Softw. Eng. Notes 32.5 (2007), pages 1–3, doi: 10.1145/1290993.1290997.

[5] R. Pitkänen and P. Selonen "A UML Profile for Executable and Incremental Specification-Level Modeling," «UML» 2004 — The Unified Modeling Language. Modeling Languages and Applications Vol 3273, Lecture Notes in Computer Science, pages 158-172 , doi 10.1007/978-3-540-30187-5_12

[6] Q. Wei, G. Danwei, X. Yaohong, F. Jingtao, H. Cheng, and J. Zhengang. "Research on software development process conjunction of scrum and UML modeling," Proceedings - 2014 4th International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC 2014), 2014, pages 978–982, doi: 10.1109/IMCCC. 2014.206.