SUGISAKA MASANORI

Journal of Advances in Artificial Life Robotics
Vol. 4(1); June (2023), pp. 35–40
ON LINE ISSN 2435-8061; ISSN-L 2435-8061
https://alife-robotics.org/jallr.html

Research Article

Prototype of the Framework CATdd to Support Continuous Development in Test Driven Development

Takeaki Miyashita[1], Tetsuro Katayama[1], Yoshihiro Kita[2], Hisaaki Yamaba[1], Kentaro Aburada[1], Naonobu Okazaki[1]

[1]*Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki, Gakuen-kibanadai nishi, Miyazaki, 889-2192, Japan*

[2]*Department of Information Security, Faculty of Information Systems, Siebold Campus, University of Nagasaki, 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki, 851-2195, Japan*

ARTICLE INFO

Article History

Received 25 Nobember 2022
Accepted 04 September 2023

Keywords

Source code generation
Test driven development (TDD)
Large language model (LLM)

ABSTRACT

TDD is a development methodology to improve software quality. In this study, we prototype the framework CATdd (Continuous Automated Test Driven Development), which supports continuous development with TDD. CATdd reduces the time of implementation step in TDD by generating new source code that passes tests using LLM. CATdd maintains consistency with other source code in the project by using existing source code and related source code to prompt. As a result of the evaluation experiments, the time for the implementation step in TDD was reduced by 94% for a simple task and by 56% for another more complex task. In conclusion, CATdd is useful to support continuous development.

© 2022 *The Author*. Published by Sugisaka Masanori at ALife Robotics Corporation Ltd.
This is an open access article distributed under the CC BY-NC 4.0 license
(http://creativecommons.org/licenses/by-nc/4.0/).

1. Introduction

Software quality and development efficiency have become more important [1]. Test Driven Development (TDD) is a development methodology to improve software quality [2]. In TDD, developers repeat a series of test design step, implementation step, and refactoring step to make test cases and source code better. By reducing the time of the implementation step in TDD, developers can focus more on test [3].

To improve the efficiency of software development, source code generation using the Large Language Model (LLM) has been studied [4], [5]. However, source code generated by LLM may have inconsistencies with other source code in the project. Examples of inconsistencies are duplication of processing between source codes, mismatch of arguments in function calls, and mismatch of coding styles. In addition, if the LLM ignores refactoring by the developer and overwrites the source code, the developer will have to repeat the refactoring. Such inconsistencies lead to increased development time and bugs in continuous development. Hence, to support continuous development, it is necessary to resolve such inconsistencies and maintain consistency in the project.

In this study, we prototype the framework CATdd (Continuous Automated Test Driven Development), which supports continuous development with TDD. CATdd reduces the time of implementation step in TDD by generating new source code that passes tests using LLM. CATdd maintains consistency with other source code in the project by using existing source code and related source code to prompt. In this prototype, we use python to implement CATdd. CATdd only supports a project that is implemented in C++ and runs tests in GoogleTest.

Corresponding author's E-mail: kat@cs.miyazaki-u.ac.jp, miyashita@earth.cs.miyazaki-u.ac.jp, kita@sun.ac.jp, yamaba@cs.miyazaki-u.ac.jp, aburada@cs.miyazaki-u.ac.jp, oka@cs.miyazaki-u.ac.jp

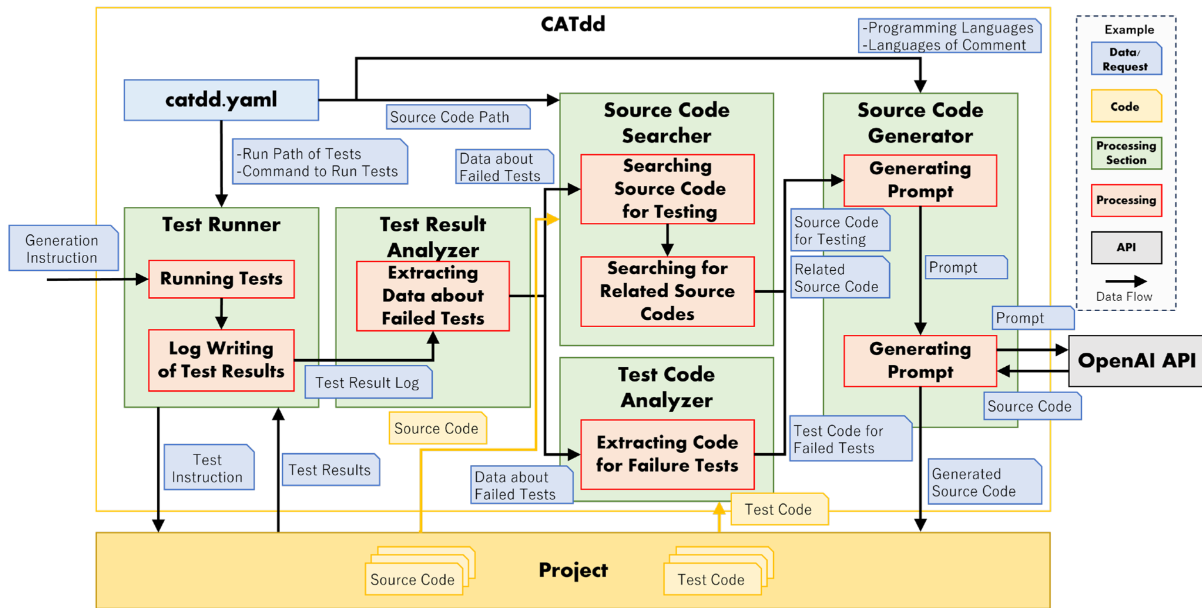

Fig. 1 The Structure of CATdd

2. CATdd

CATdd is a framework to support continuous software development with TDD. CATdd runs tests on the target project and generates source code that can pass the tests based on the source code that failed the tests, maintaining the consistency in the project by using LLM. In this process, CATdd refers to another source code that is loaded by the source code to be tested as a related source code. To generate source code with CATdd, the developer must pre-set 5 data in catdd.yaml: the source code path, the test execution path, the test execution command, the programming language used in the project, and the language of comments when generating source code.

3. Implementation

The structure of CATdd is shown in Fig. 1. As shown in Fig. 1, CATdd has 5 processing sections: Test Runner, Test Result Analyzer, Source Code Searcher, Test Code Analyzer, and Source Code Generator. The behavior of each processing section is described below.

3.1. *Test Runner*

Test Runner runs tests on the project supported by CATdd using the test execution path and test execution commands in catdd.yaml, and redirects the test results to a log file. The log file of the test results is sent to the Test Result Analyzer.

3.2. *Test Result Analyzer*

Test Result Analyzer extracts data about failed test cases from the test results received from the Test Runner. Test Result Analyzer extracts failed test cases that match the pre-defined regular expression patterns for assertion errors and the 3 errors: “No such file”, “has no member”, and “undefined”. The extracted test cases data is sent to Source Code Searcher and Test Code Analyzer.

3.3. *Source Code Searcher*

Source Code Searcher searches for source files based on the class names that failed to pass the test from the information received from the Test Result Analyzer. If a source file is found, the header file that the source file includes is also searched. The source code of the discovered file is passed to the Source Code Generator. In this prototype, only the header file of the class to be tested was the target of the search. If no files are found, empty character is sent to the Source Code Generator.

```
Implement the source code for class {Class Name of Failed Test}
that satisfies the following conditions using {Programing
Language} with {Language of Comments} comments.
- Only the source file must be implemented out of the two files,
header file and source file.
- Satisfy all of the following test cases. However, it is not
necessary to run the tests.
{Test Code for Failed Test}
- Based on the following source code.
{Existing Source Code}
- Based on the following header file.
{Related Source Code}
```

Fig. 2. The Syntax of Prompt

```
TEST(calculateMileageTest, calculateMilage)
{
  double radius = 50;
  double rightAngle = 10;
  double leftAngle = 20;
  double rightWheel = 2 * rightAngle * radius * M PI / 360;
  double leftWheel = 2 * leftAngle * radius * M PI / 360;
  double expected = (rightWheel + leftWheel) / 2;
  double actual = Mileage::calculateMileage(rightAngle,
leftAngle);
  EXPECT DOUBLE EQ(expected, actual);
}
```

Fig. 3. Part of the Test Code

3.4. *Test Code Analyzer*

Test Code Analyzer extracts the test code for the failed test cases based on the test code line numbers received from the Test Result Analyzer and sends it to the Source Code Generator.

3.5. *Source Code Generator*

Source Code Generator generates prompts for source code that pass testing and maintain consistency with the related source code, based on code received from Source Code Searcher and Test Code Analyzer. Then, Source Code Generator sends the generated prompts to the OpenAI API [6] for generating the source code. This prototype uses the text-davinci-003 model. Source Code Generator generates 2 files, a source file (.cpp) and a header file (.h), for the target of the failed test.

The syntax of the prompt is shown in Fig. 2. The data composing the prompt, the programming language and the language of comments, are taken from the configuration file catdd.yaml, the class name of the failed test target, the existing source code, and the related source code are taken from the Source Code Searcher, and the test code of the failed test is taken from the Test Code Analyzer.

The sending the prompt to the OpenAI API has a problem, the request is rejected if the number of tokens is over the limit [6]. To solve the problem, Source Code Generator ranks the data that makes the prompt in order of priority and reduces them in the order of decreasing priority. The data is reduced in the following order: comments in the relevant source code, comments in the source code that failed to pass existing tests, failed test cases, relevant source code, source code that failed to pass existing tests. In the reducing data for failed test cases, at least one of the failed test cases is included in the prompt. When the number of tokens is below the limit, Source Code Generator generates the prompt and sends it to the OpenAI API for source code generation. Finally, Source Code Generator writes the generated source code to a file on the project. If the number of tokens is over the limit after reducing all the data, Source Code Generator notifies the developer and exits the process.

4. Application Example

To confirm that the prototype CATdd works as expected, using an application example. In an empty project supported by CATdd, test code and header files were manually described for the Mileage class that calculates the movement distance from the rotation angles of both wheels, and generated source code by CATdd. A part of the test code and the header file used to generate the source code are shown in Fig. 3 and Fig. 4, and the source file generated by CATdd is shown in Fig. 5. After source code generation, the project passed all tests. This confirms that CATdd can generate source code that can pass tests. Also, Fig. 5 shows that CATdd implemented the private function "calculateWheelMileage()", which is not described in the test code, by using the header file shown in Fig. 4 for the source code generation. This confirms that CATdd can maintain consistency with other source code in the project by using source code related to the target of generation as input.

```
#ifndef MILEAGE_H
#define MILEAGE_H
#include <cmath>
class Mileage {
 public:
  /**
   * @brief Calculate mileage of both wheel
   * @param rightAngle Right motor angle [deg]
   * @param leftAngle Left motor angle [deg]
   * @return Mileage of both wheel [mm]
   */
  static double calculateMileage(int rightAngle, int leftAngle);

 private:
  static const double RADIUS;
  Mileage();

  /**
   * @brief Calculate mileage of a wheel
   * @param angle Motor angle of a wheel [deg]
   * @return Mileage of a wheel [mm]
   */
  static double calculateWheelMileage(int angle);
};
#endif
```

Fig. 4. Described Header File

```
// Mileage.cpp
// Implementation of Mileage class
#include "Mileage.h"
// Constructor
Mileage::Mileage() {}
double Mileage::calculateMileage(int rightAngle, int leftAngle) {
   double rightWheel = calculateWheelMileage(rightAngle);
   double leftWheel = calculateWheelMileage(leftAngle);
   return(rightWheel + leftWheel) / 2;
}
double Mileage::calculateWheelMileage(int angle) {
   return 2 * angle * RADIUS * M_PI / 360;
}
```

Fig. 5. Generated Source Code By CATd

5. Evaluation

We evaluate the effective of the time of implementation step in TDD by comparing the implementation with and without CATdd in the experiment. 7 students majoring in computer science and engineering, 4 graduate students and 3 undergraduates, participated in the experiment as subjects. They solved 2 tasks using TDD. 4 subjects used CATdd for only the first task, and other subjects used CATdd for only the second task.

Table 1. The Average Implementation Time per Person

Method	1st Task	2nd Task
Manual	89m33s	101m16s
CATdd	**5m19ss**	**44m21s**

The first task is to implement a Mileage class that calculates the movement distance from the rotation angles of both wheels. This task is simple and can be implemented with about 30 lines of source code for 5 test cases. The second task is to implement the member function calculatePid() of the Pid class that executes PID control. This task is more complex and can be implemented with about 20 lines of additional source code, for 8 test cases and about 50 lines of existing source code. The test code was created by the experimenter and all test cases were commented out.

In the experiment, subjects executed the following steps: uncomment a test case, run the test to check the results, implement source code that pass the test, refactor if necessary, and repeat these steps until the task is completed. Additionally, after completing both tasks, they submitted feedback on using CATdd.

5.1. *Evaluation of Implementation Time*

The average implementation time per person for each combination of task and implementation method is shown in Table 1. Table 1 shows that CATdd reduced the time of implementation by 84m14s (94%) for the first task and by 56m55s (56%) for the second task. This indicates that CATdd is useful for reducing time of implementation step in TDD.

5.2. *Evaluation of the Generated Source Code*

During the experiment, CATdd generated source code that passed most tests. On the other hand, CATdd generated source code that failed to consider the pattern and did not pass the test, even though there was a test case for division by zero. Also, during the experiment, the generated source code was sometimes not source code or contained errors that CATdd did not support. Generating incorrect source code increases developer effort and development time. To improve the quality of generated source code, it is necessary to increase the quantity and quality of information used for source code generation.

5.3. *Evaluation of Usefulness by Questionnaire*

Subjects answered what was good and bad about CATdd after the task was completed.

As good points, 5 of the 7 subjects mentioned that the task was easier, and 3 mentioned that they were able to implement it faster. In addition, the comments generated automatically were useful, and the number of implementation errors was reduced. On the other hand, as bad points, 3 of the 7 subjects were able to implement without understanding the source code generated automatically, and 2 subjects found it difficult to understand what changes had been made due to the automatic generation. In addition, they also mentioned the time and effort required to delete unnecessary code, change variable names, etc., which is not required in manual development.

From these answers, it can be said that CATdd is useful in reducing development time in TDD.

5.4. *Related Research*

One research that automatically generates source code is CodeT developed by Chen et al [4]. CodeT generates source code using natural language requests as input. In addition, it also generates test code and evaluates the generated source code when it generates source code. However, CodeT needs to re-generate source code with new requirements as input when the requirements are changed. On the other hand, CATdd uses the existing source code as input and can generate new source code maintaining the changes made by the developer's refactoring. Since minor modifications by the developer are retained in the next source code generation, CATdd can support continuous development.

One research of APR (Automated Program Repair), which automatically corrects test failures and errors, is the work of Chunqiu et al [5]. In this study, they proposed a conversational APR that alternates between generating patches and executing tests using LLMs. Conversational APR generates new patches by taking as input a program containing bugs, test cases, and previously generated patches and test results for those patches. Repeated input of previously generated patches and test results to the model prevents the generation of patches that have already been generated. However, since only one program is used as input, it does not support processes that can be realized by multiple programs. On the other hand, CATdd uses the related source code loaded in the source code as input for source code generation, in addition to the failed tests and the source code to be tested. By using the related source code, CATdd improves the possibility of generating source code with consistency within the project.

6. Conclusion

We have prototyped the framework CATdd, which supports continuous development with TDD. Using the application example, we have confirmed that CATdd can maintain consistency with other source code in the project. As a result of the evaluation experiments, the time for the implementation step in TDD was reduced by 94% for simple task and by 56% for more complex task. In addition, the questionnaire showed that CATdd is useful in reducing implementation time and human error. In conclusion, CATdd is useful to support continuous development in TDD.

Our future tasks are shown below.

- Improve quality of generated source code
- Extend range of related source code support
- Guide to automatically generated code

References

1. "BBC: What went wrong inside Boeing's cockpit?" https://www.bbc.co.uk/news/extra/IFtb42kkNv/boeing-two-deadly-crashes#group-System-failurezAFs52mVNd (Accessed 2023-9-3).
2. Rafique, Y., and Miˇsi´c, V., "The Effects of Test-DrivenDevelopment on External Quality and Productivity: AMeta-Analysis", IEEE Transactions on Software Engineering, Vol.39, No.6, pp.835-856, 2013.
3. Anwer, F., Aftab, S., Waheed, U., et al., "Agile SoftwareDevelopment Models TDD, FDD, DSDM, and CrystalMethods: A Survey", International Journal of MultidisciplinarySciences and Engineering, Vol.8, No.2 pp.1-10, 2017.
4. Chen, B., Zhang, F., Nguyen, A., et al., "CodeT: Code Generation with Generated Tests", The Eleventh International Conference on Learning Representations 2023 poster, 19p. 2023.
5. Xia, C.S. and Zhang, L., "Conversational Automated Program Repair", arXiv:2301.13246, 11p, 2023
6. "OpenAI: OpenAI API Documentation", https://platform.openai.com/docs/introduction, (Accessed 2023-7-31)

Authors Introduction

Mr. Takeaki Miyashita

He received the Bachelor's degree in engineering (computer science and systems engineering) from the University of Miyazaki, Japan in 2022. He is currently a Master's student in Graduate School of Engineering at the University of Miyazaki, Japan. His research interests software testing and software quality.

Dr. Tetsuro Katayama

He received a Ph.D. degree in engineering from Kyushu University, Fukuoka, Japan, in 1996. From 1996 to 2000, he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000 he has been an Associate Professor at the Faculty of Engineering, Miyazaki University, Japan. He is currently a Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing and quality. He is a member of the IPSJ, IEICE, and JSSST.

Dr. Yoshihiro Kita

He received a Ph.D. degree in systems engineering from the University of Miyazaki, Japan, in 2011. He is currently an Associate Professor with the Faculty of Information Systems, University of Nagasaki, Japan. His research interests include software testing and biometrics authentication.

Dr. Hisaaki Yamaba

He received the B.S. and M.S. degrees in chemical engineering from the Tokyo Institute of Technology, Japan, in 1988 and 1990, respectively, and the Ph D. degree in systems engineering from the University of Miyazaki, Japan in 2011. He is currently an Assistant Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include network security and user authentication. He is a member of SICE and SCEJ.

Dr. Kentaro Aburada

He received the B.S., M.S, and Ph.D. degrees in computer science and system engineering from the University of Miyazaki, Japan, in 2003, 2005, and 2009, respectively. He is currently an Associate Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include computer networks and security. He is a member of IPSJ and IEICE.

Dr. Naonobu Okazaki

He received his B.S, M.S., and Ph.D. degrees in electrical and communication engineering from Tohoku University, Japan, in 1986, 1988 and 1992, respectively. He joined the Information Technology Research and Development Center, Mitsubishi Electric Corporation in 1991. He is currently a Professor with the Faculty of Engineering, University of Miyazaki since 2002. His research interests include mobile network and network security. He is a member of IPSJ, IEICE and IEEE.