Clone Merge—An Eclipse plugin to abstract near-clone C++ methods

Krishna Narasimhan
Institut für Informatik
Goethe University Frankfurt
krishna.nm86@gmail.com

***Abstract*—Software clones are prevalent. In the work of Laguë et al. [2], they observe that 6.4% and 7.5% of the source code in different versions of a large, mature code base are clones. The work of Baxter et al. [1] reports even higher numbers, sometimes exceeding 25%. We consider the prevalence of such near miss clones to be strong indicators that copy-paste-modify is a wide-spread development methodology. Even though clones are prevalent, they are a significant development headache. Specially, if bugs arise in one of the clones, they need to be fixed in all of the clones. This problem is acknowledged in the work of Juergens et al. [4] who say in their work that "*cloning can be a substantial problem during development and maintenance*", since "*inconsistent clones constitute a major source of faults*". A similar concern is raised in practitioner literature [3] suggesting that clones should be removed in some form or the other. We present a tool that can be installed as a plugin to Eclipse CDT, the development environment for C/C++. The research prototype comes with a refactoring option called "Copy Paste merge" refactoring, which is available as a menu option in the modified version of the Eclipse CDT.**

I. Overview

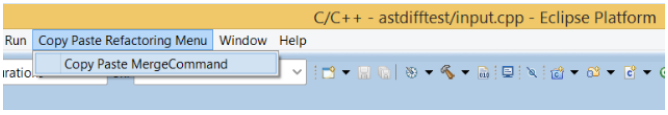

Fig. 1. Eclipse refactoring option

The tool is inspired by the approach presented by Krishna Narasimhan and Christoph Reichenbach in their paper titled "Copy and Paste Redeemed" to appear in the same conference's main track, ASE 2015. The overview of the approach is simple. Each method definition suspected to be copy pasted is provided as an input in the AST form to a a merge function. We use the merge function to identify the nodes that are unique to each subset of the ASTs. Using these unique nodes and the ASTs they come from, merge points are generated. For each merge point, a resolution pattern is decided based on the type of the node. A resolution pattern is simply a code transformation pattern that determines the appropriate way to merge nodes. For examples, nodes containing constants can be merged using an extra parameter to the function. The current version of the prototype works on a single input file that contains all the clone groups marked with pragma annotations that specify which are the methods that are to be used as input for the refactoring. The clone groups can be of any size in terms of number of AST nodes. A *clone group* is a group of methods which the developers consider to be near-clones and believes should be abstracted. The input are to be placed in a special file called "input.cpp" inside a project called "astdifftest". For more detailed instructions, refer to the website. http://krishnanm86.wix.com/clonerge. The refactoring creates or updates a file called "input.cppmergedoutput.cpp" which contains the clone groups merged into one function called "mergedFunction". The function definitions of the clone groups are replaced with calls to the merged function with parameters and/or template parameters helping the merged function as to how it decides the merge process. Currently, four kinds of merges are supported:

1) Differences in Constants are resolved using an extra parameter based on the type of the constant
2) Differences in Types are resolved using a template parameter.
3) Differences in statements are resolved using a Switch Statement
4) Differences in names (such as field names/ variable names) are resolved using extra parameter. If the name happens as part of a left hand side of an assignment, corresponding pointer operations are added to the caller and callee.

II. Scenarios

Here, let us see a few real scenarios where our tool might be used. One for each type of difference:

A. Differences in Constants

Consider **Figure 2** which is the input of a clone group with differences of constants. For each of the constants in (action, module and clientId), an extra parameter is introduced to the merged function and that parameter is placed in the position of the constants. And each of the calling functions, pass their respective constants which can be seen in the output in **3**.

B. Differences in Field names and Statements

Consider **Figure 4** which is the input of a clone group with differences of statements of calls to fn1 and fn2 and also differences in field access of (x,y) and (cellDiv, cellMod). For the statement difference, a switch statement is introduced and an extra parameter denoting which function to switch between. For the field accesses an extra parameter of unknown template type is introduced and for (x,y), a pointer operation is introduced because it happens in the LValue. The output can be seen in **5**.

```
//#pragma ast 1
Handle<Value> Connection::GetModule (Local<String> property,
                                     const AccessorInfo& info)
{
  HandleScope scope;
  Connection* njsConn = ObjectWrap::Unwrap<Connection>(info.Holder());
  if(!njsConn->isValid_)
    msg = NJSMessages::getErrorMsg(errInvalidConnection);
  else
    msg = NJSMessages::getErrorMsg(errWriteOnly, "module1");
  NJS_SET_EXCEPTION(msg.c_str(), (int) msg.length());
  return Undefined();
}

//#pragma ast 2
Handle<Value> Connection::GetClientId(Local<String> property,
                                      const AccessorInfo& info)
{
  HandleScope scope;
  Connection* njsConn = ObjectWrap::Unwrap<Connection>(info.Holder());
  if(!njsConn->isValid_)
    msg = NJSMessages::getErrorMsg(errInvalidConnection);
  else
    msg = NJSMessages::getErrorMsg(errWriteOnly, "clientId");
  NJS_SET_EXCEPTION(msg.c_str(), (int) msg.length());
  return Undefined();
}

//#pragma ast 3
Handle<Value> Connection::GetAction(Local<String> property,
                                    const AccessorInfo& info)
{
  HandleScope scope;
  Connection* njsConn = ObjectWrap::Unwrap<Connection>(info.Holder());
  if(!njsConn->isValid_)
    msg = NJSMessages::getErrorMsg(errInvalidConnection);
  else
    msg = NJSMessages::getErrorMsg(errWriteOnly, "action");
  NJS_SET_EXCEPTION(msg.c_str(), (int) msg.length());
  return Undefined();
}
```

Fig. 2. Constant Difference - Input

III. Relation to other tools

As far as we are aware, this is the first tool that breaks the barriers of using only grammatical information while performing merges of near clones. There are refactoring options that allow for extracting methods from common code. But they are very trivial and work on exact differences. They are not able to perform such type and other sophisticated program analysis that we perform in order to find the points of abstraction and performing an abstraction using abstraction patterns like using a Template and Switch patterns. For a future version of the tool, we plan to introduce a User selection, where the user can choose between abstraction patterns. For Example, using an If conditional branch instead of a Switch. Or using a global field instead of an extra parameter. This is easily possible because of the way the tool is built, using merge points and resolution patterns which is described in a complementary paper submitted to ASE 2015 titled "Copy and Paste redeemed". In the terminology of Koschke et al. [5]), they mention three types of clones, with Type-3 being the most evolved and sophisticated of clones which have considerable parts of their ASTs differing. Perhaps the most closely related clone management approach to our algorithm is Cedar [7], which targets Java and relies on Eclipse refactorings for abstraction. Our approach is more general and does not rely on existing refactorings like Extract Method. Unlike our approach, Cedar is limited to Type-2 clones.

IV. Impact to Industry

To the best of our knowledge, ours is the only work to support merging the common Type-3 clones (inexact clones) in a wide variety of cases. As Roy et al. [6] note, Type-3 clones are particularly common and frequently evolve out of Type-1 and 2 clones. This already sets our tool potentially in the forefront of industry application. In order to test the quality of abstractions performed by our tool, we checked out popular GitHub repositories and abstracted a few of their clone groups using our tool. During the month of February 2015, we picked the top trending repositories from GitHub, gathered

```
#pragma ast 1

Handle<Value> mergedFunction(Local<String> property, const AccessorInfo& info,
        wchar_t id1333123[]) {
    HandleScope scope;
    Connection* njsConn = ObjectWrap::Unwrap < Connection > (info.Holder());
    if (!njsConn->isValid_)
        msg = NJSMessages::getErrorMsg(errInvalidConnection);
    else
        msg = NJSMessages::getErrorMsg(errWriteOnly, id1333123);

    NJS_SET_EXCEPTION(msg.c_str(), (int) (msg.length()));
    return Undefined();
}

Handle<Value> Connection::GetModule (Local<String> property,
                                     const AccessorInfo& info)
 {
     return  mergedFunction(property, info, "module1");
}

#pragma ast 2
Handle<Value> Connection::GetClientId(Local<String> property,
                                      const AccessorInfo& info)
 {
     return  mergedFunction(property, info, "clientId");
}

#pragma ast 3
Handle<Value> Connection::GetAction(Local<String> property,
                                      const AccessorInfo& info)
 {
     return  mergedFunction(property, info, "action");
}
```

Fig. 3. Constant Difference - Output

```
#pragma ast 1
Type typeDiv(Type t1, Type t2) {
  fn1();
  if (auto t = eval_const_divmod(t1, t2, cellDiv)) return *t;
  x = 10;
  return TInitPrim;
}

#pragma ast 2
Type typeMod(Type t1, Type t2) {
    fn2();
  if (auto t = eval_const_divmod(t1, t2, cellMod)) return *t;
  y = 10;
  return TInitPrim;
}
```

Fig. 4. Statement and Field Name- Input

```
#pragma ast 1

template<template<> class T13311, template<> class T13212214>
Type mergedFunction(Type t1, Type t2, T13311* id13311, T13212214 id13212214,
        int funcName131) {
    switch (funcName131) {
    case 2:
        fn2();
        break;
    case 1:
        fn1();
        break;
    }
    if (auto t = eval_const_divmod(t1, t2, id13212214))
        return *t;

    *id13311 = 10;
    return TInitPrim;
}

Type typeDiv(Type t1, Type t2) {
    return mergedFunction(t1, t2, &x, cellDiv, 1);
}

#pragma ast 2
Type typeMod(Type t1, Type t2) {
    return mergedFunction(t1, t2, &y, cellMod, 2);
}
```

Fig. 5. Statement and Field Name- Output

clone groups using a simple Clone Detector implemented by ourselves. We picked random clone groups, abstracted them using our tool and submitted 10 Clone Groups abstracted as pull requests. Out of these, 9 were merged back into the code. We even got positive feedback from many of the Maintainers including the ones from Google Protobuf and Oracle Nodedb repositories. This indicates that people in the industry prefer to have such abstractions in their code bases. This evaluation also indicated that there is a common prevalence of copy-paste-modify method of extending functionality which seems to be easier. So, our tool can allow industry users to apply copy-paste mode of refactoring which is easy and then use our tool to abstract them into a preferred form of the final code. Our pull requests can be found in **Figure 6**

V. Website

Here is a website, which has complete information about test inputs, a sample video demonstrating a run of the tool and also a downloadable version of the Eclipse CDT with the refactoring pre installed and instructions on how to install and use it.

http://krishnanm86.wix.com/clonerge

The link to the download of the tool is :

https://www.dropbox.com/l/EqsUNLh6oQagDo7HmDVcir

The link to a video demonstration of the tool is : http://youtu.be/vm8s0-TM0tY?hd=1

References

[1] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 368–, Washington, DC, USA, 1998. IEEE Computer Society.

[2] Bruno Laguë, Daniel Proulx, Ettore M. Merlo, Jean Mayrand, and John Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In *Proc. Int'l Conf. Software Maintenance (ICSM)*, pages 314–321. IEEE Computer Society Press, 1997.

[3] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[4] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 485–495, May 2009.

[5] Rainer Koschke, Raimar Falke, and Pierre Frenzel. Clone detection using abstract syntax suffix trees. In *Proceedings of the 13th Working Conference on Reverse Engineering*, WCRE '06, pages 253–262, Washington, DC, USA, 2006. IEEE Computer Society.

[6] C.K. Roy, K.A. Schneider, and D.E. Perry. Understanding the evolution of type-3 clones: An exploratory study. MSR 2013.

[7] Robert Tairas and Jeff Gray. Increasing clone maintenance support by unifying clone detection and refactoring activities. *Inf. Softw. Technol.*, 54(12):1297–1307, December 2012.

Fig. 6. Repositories with their pull request URLs. Each clone group represents one abstraction.We encourage the readers who choose to look at the pull requests to go through the comments. Although some of the pull requests don't explicitly have the status as "merged", like with the OracleDB and the MongoDB repositories, the codes have actually been merged, indicated by the comments of the Maintainers.

Repository	**Clone Groups**	**Status**	**URL**
oracle/node-oracledb	3	Accepted	• https://github.com/oracle/node-oracledb/pull/28
mongodb/mongo	2	Accepted	• https://github.com/mongodb/mongo/pull/927 • https://github.com/mongodb/mongo/pull/928
rethinkdb/rethinkdb	2	Accepted	• https://github.com/rethinkdb/rethinkdb/pull/3820 • https://github.com/rethinkdb/rethinkdb/pull/3818
cocos2d/cocos2d-x	2	Accepted	• https://github.com/cocos2d/cocos2d-x/pull/10539 • https://github.com/cocos2d/cocos2d-x/pull/10546
ideawu/ssdb	1	Rejected	• https://github.com/ideawu/ssdb/pull/609