Automated Program Repair

Opportunities, Challenges, Advances

Chris Timperley
About me...

ctimperley@cmu.edu

Research Interests:
- Automated Program Repair
- Fault Localisation
- GI for Robotics
- Automated Test Generation

Recent Projects:
- **BugZoo**: reproducible studies of historical bugs
- **Rooibos**: language-independent, syntax-aware search and transformation.
- **Houston**: automated testing for robotics

Postdoc
Carnegie Mellon University, USA
w/ Claire Le Goues

MEng (2013), PhD (2017)
University of York, UK
w/ Susan Stepney
Purpose of Talk:
Challenge existing views, and identify opportunities.
Genetic Programming to modify existing programs, rather than building them from scratch.

Demonstrates concept of automated program repair.

Evolutionary Repair of Faulty Software
Andrea Arcuri
The School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TJ, UK. Email: a.arcuri@cs.bham.ac.uk

Abstract

Testing and fault localization are very expensive software engineering tasks that have been tried to be automated. Although many successful techniques have been designed, the actual change of the code for fixing the discovered faults is still a human-only task. Even in the cited case in which automated tools could tell us exactly where the location of a fault is, it is not always trivial how to fix the code. In this paper we analyze the possibility of automating the complex task of fixing faults. We propose to model this task as a search problem, and hence to use for example evolutionary algorithms to solve it. We then discuss for potential of this approach and how its current limits can be addressed in the future. This task is extremely challenging and hardly accepted in literature. Hence, this paper only covers an initial investigation and gives directions for future work. A research prototype called JAFF and a case study are presented to give first validation of this approach.


1 Introduction

Software testing is used to reveal the presence of faults in computer programs [50]. Even if no fault is found, testing cannot guarantee that the software is fault-free. However, testing can be used to increase our confidence in the software reliability. Unfortunately, testing is expensive, time consuming and tedious. It is estimated that testing requires around 50% of the total cost of software development [14]. This is the reason why there has been a lot of effort spent to automate this expensive software engineering task.

Even if an optimal automated system for doing software testing existed, we still need to know where the faults are located, that is in order to be able to fix them. Automated techniques can help the tester in this task [28, 65, 78].

Although in some cases it is possible to automatically locate the faults, there is still the need to modify the code to remove the faults. Is it possible to automate the task of fixing faults? This would be the natural next step if we seek a full automation of software engineering. And it would be particularly helpful in the cases of complex software in which, although the faulty part of code can be identified, it is difficult to provide a patch for the fault. This would also be a step forward to achieve corporate visions like for example IBM’s autonomic Computing [10].

There has been work on fixing code automatically (e.g., [63, 61, 68, 25]). Unfortunately, in that work there are heavy constraints on the type of modifications that can be automatically done on the source code. Hence, only limited classes of faults can be addressed. The reason for putting these constraints is that there are infinite ways to do modifications on a program, and checking all of them is impossible.

Evolutionary repair of faulty software.
Andrea Arcuri. 2011.
GenProg demonstrates program repair on real-world C programs

Automatically Finding Patches Using Genetic Programming
Westley Weimer, ThanVu Nguyen, Claire Le Goues, Stephanie Forrest.
Fault Localisation

LOCALISED CODE

Patch Generation

FIXED CODE

Program Analysis

2009–
Where are the program repair bots?

Or, why aren’t we all filthy rich yet?
9

How to Design a Program Repair Bot? Insights from the Repairnator Project.
Simon Urli, Zhongxing Yu, Lionel Seinturier, and Martin Monperrus.

they exist!

(note: we found out about them a few weeks ago)
How do we deploy?
What do developers need?

- Easy Integration
- Timeliness
- Bug Information
- Patches
The big assumption: The existence of tests

requires test cases
Software engineering has changed since APR was introduced.
[triggers CI build]

[pushes results]
requires test cases
Coveralls commented on Dec 8, 2017

Coverage increased (-0.7%) to 96.774% when pulling f64af5a on commits-to-function into 9b0556a on master.
What are the challenges?
Challenges

- Patch Quality
- Scalability
- Expressiveness
Challenges: Patch Quality

BUGGY

```c
max = 0;
if(a > b)
    max = a;
if(b > a)
    max = b;
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

CORRECT

```c
max = a;
if(a > b)
    max = a;
if(b > a)
    max = b;
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

PLAUSIBLE

```c
max = 7;
if(a > b)
    largest = a;
if(b > a)
    largest = b;
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
Not all good patches are correct.
“In many cases the Kali patch cleanly identifies the exact functionality and location that the developer patch modifies”

“The Kali and developer patches typically modify common functionality and variables.”

“Bug reports also accompanied by [machine-generated] patches were three times as likely to be addressed as standard bug reports.”
Correctness is a major challenge, but overfitted patches can still be useful.
Challenges

Patch Quality  Scalability  Expressiveness
Challenges: Expressiveness vs. Scalability

Expressiveness

Allows the program to be changed in a greater number of ways, increasing the odds of finding a modification that produces a repair.

- larger corpus of fix ingredients
- wider set of program transformations
- granular modifications to the program
Challenges: Expressiveness vs. Scalability

Time taken to discover a patch is a function of:

- patch size
- program size
- expressiveness
- ...

Scalability
How can we make APR both scalable and expressive?
Observation:
APR inherited most of its technologies.
Program Analysis

Fault Localisation

Patch Generation

compiler optimisation
test suite prioritisation
abstract syntax trees
concolic execution
specification mining
...

genetic algorithms
delta-debugging minimisation
random search
program synthesis
metaprogramming
mutation testing
...

BUGGY CODE

LOCALISED CODE

FIXED CODE
Abstract Syntax Tree

Patch Representation

APPEND 12 3
SWAP 5 6
DELETE 4
Fault Localisation

Program Analysis

Patch Generation

compiler optimisation
test suite prioritisation
abstract syntax trees
concolic execution
specification mining

... genetic algorithms
delta-debugging minimisation
random search
program synthesis
metaprogramming
mutation testing

...
Everyone is *still* using spectrum-based fault localisation!

- **2017**: ssFix, Repairnator, NOPOL, ...
- **2016**: History-Driven Program Repair, ...
- **2015**: Angelix, SearchRepair, ...
- **2014**: Astor, RSRepair, ...
- **2013**: SemFix, ...
- **2011**: AE, ...
- ...
- **2009**: GenProg
Recap: Spectra-Based Fault Localisation

```c
int year, days;
year = 1980;
days = atoi(argv[1]);
while (days > 365) {
    if (isLeapYear(year)){
        if (days > 366) {
            days -= 366;
            year += 1;
        } else {
        }
    } else {
        days -= 365;
        year += 1;
    }
} return year;
```
\[
\frac{e_f}{e_f + n_f + e_p}
\]
\[
\frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}
\]
\[
\frac{e_f}{e_f + n_f} - \frac{e_p}{e_p + n_p}
\]
\[
0, \quad \text{if } e_f + e_p = 0
\]
\[
1.0, \quad \text{if } e_f > 0 \land e_p = 0
\]
\[
0.1, \quad \text{otherwise}
\]
\[
\frac{e_f}{e_f + n_f}
\]
\[
\frac{e_p}{e_p + n_p} + \frac{e_f}{e_f + n_f}
\]
Assumption:
several failing test cases (6 is optimal).

Reality:
usually one failing test.

Takes > 12 hours to run GenProg and SearchRepair.
Can we tailor fault localisation to CI-based program repair?