The Living Review on Automated Program Repair

Martin Monperrus


Concept This paper is a living review on automatic program repair\(^1\). Compared to a traditional survey, a living review evolves over time. I use a concise bullet-list style meant to be easily accessible by the greatest number of readers, in particular students and practitioners. Within a section, all papers are ordered in a reverse chronological order, so as to easily get the research timeline. The references are sorted chronologically and years are explicitly stated inline to easily grasp the most recent references.

Inclusion criterion The selection criterion is that the considered papers must be about automatic repair with some kind of patch generation (runtime repair without patch generation is excluded\(^2\)). The cited papers must contain a reasonable amount of material (at least 4 double-column pages). There is no restriction about whether the paper has been formally peer-reviewed or not.

Originality Compared to formal surveys [122, 117], this living review contains very recent references and continues to evolve. It uses a bullet-list concise style that is not typical academic writing.

Feedback I would be very happy to read from you if you spot a mistake, a confusing statement or a missing paper: martin.monperrus@csc.kth.se.

Version history

- May 2019: version with 209 references
- April 2019: version with 205 references
- March 2019: version with 200 references
- February 2019: version with 193 references
- December 2018: version with 175 references

Citation This living review can be cited as: “Living Review on Automated Program Repair”, Martin Monperrus, Technical Report HAL # hal-01956501, 2018.

\(\text{\cite{repair-living-review}}\)

\(^{1}\text{https://en.wikipedia.org/wiki/Living_review}\)

\(^{2}\text{the scope of my previous survey [122] was larger, it also discussed runtime repair}\)
## Contents

1 Program Repair of Dynamic Errors .................................................. 3
   1.1 Using Tests ........................................................................... 3
   1.2 Using Crashes .................................................................... 5
   1.3 Using a Reference Implementation ....................................... 5
   1.4 Using Contracts .................................................................. 6
   1.5 Patch Generation in Production .......................................... 7

2 Program Repair of Static Errors ....................................................... 7
   2.1 Static Warnings ................................................................... 7
   2.2 Compiler Errors .................................................................. 8

3 Empirical Studies for Program Repair ............................................. 8

4 Targeted Repair ............................................................................. 11
   4.1 Test Repair .......................................................................... 11
   4.2 Automated Repair of Concurrency errors .......................... 11
   4.3 Automated Repair of Build Scripts ..................................... 11
   4.4 Repair for the Web ............................................................. 12
   4.5 Repair of Software Models ............................................... 12
   4.6 Misc Repair Types ............................................................ 12

5 Optimization & Integration .............................................................. 13
   5.1 Driving the Search ............................................................... 13
   5.2 Improvement of the Fault Localization Step ...................... 13
   5.3 Repair Speed ...................................................................... 13
   5.4 Integration in Open-Source / Industry ................................. 14

6 Position Papers ............................................................................... 14

7 Formal Approaches to Program Repair .......................................... 14

8 Miscellaneous .................................................................................. 15
   8.1 Benchmarks ......................................................................... 15
   8.2 Automatic Hardening .......................................................... 15
   8.3 Surveys ................................................................................ 15
   8.4 Doctoral Theses ................................................................. 16
1 Program Repair of Dynamic Errors

1.1 Using Tests


- Shaping Program Repair Space with Existing Patches and Similar Code (2018) [158] selects the most similar repair ingredients that are also instances of bug fix patterns mined over past commits.


- Towards practical program repair with on-demand candidate generation (2018) [157] does repair with metaprograming as [116] in order to explore the search space of variable and literal replacement.

- CFAAR: Control Flow Alteration to Assist Repair (2018) [185] uses specific patterns to determine angelic values à la Nopol [134] (e.g., switch only the first execution of the condition).

- Context-Aware Patch Generation for Better Automated Program Repair (2018) [192] considers an ingredient-based, generate-and-validate repair loop à la Genprog, and selects the ingredients that have the most similar context according to three similarity metrics (context of the suspicious statement similar to context of the ingredient). (code)

- Practical Program Repair via Bytecode Mutation (2018) [150] revisits Schulte’s work [26] for Java bytecode and Defects4J.

- Program Repair via Direct State Manipulation (2018) [155] proposes a variation of the repair problem: find a patch such that some variables at a specific location have certain values.

- Connecting Program Synthesis and Reachability: Automatic Program Repair Using Test-Input Generation (2017) [124] creates a meta-program parametrized with parameters, encoding the search space. The symbolic solution to satisfy all test constraints is the patch.

- Contract-based Program Repair Without the Contracts (2017) Chen et al. [113] uses 5 repair templates, called schemas, with a focus on modifying the state by adding an assignment. (code)

- Precise Condition Synthesis for Program Repair (2017) Xiong et al. [133] integrate different heuristics (Github) and code analysis techniques (dependency analysis between variables) to create good conditions à la Nopol. (code)

- Leveraging syntax-related code for automated program repair (2017) Xin and Reiss [131] use Tf-Idf similarity to select ingredients in a GenProg-like loop, together with variable renaming to adapt repair ingredients. The authors have proposed an improvement of ssFix called sharpFix [207].

- ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming (2017) [138] combines 3 different techniques (patch representation, multi-objective search, method scope) to improve a GenProg-based repair loop.

- ELIXIR: Effective Object Oriented Program Repair (2017) [125] proposes 8 repair patterns à la PAR [49] to be used together with simple enumeration-based synthesis.
• **ASTOR: A Program Repair Library for Java** (2016) [109] presents the Java framework in which jGenProg [121], jKali [121], DeepRepair [130], Cardumen [170] are implemented.

• **Automated Program Repair by Using Similar Code Containing Fix Ingredients** (2016) [103] modifies RSRepair [71] in order to select the most similar repair ingredients first.

• **DynaMoth: Dynamic Code Synthesis for Automatic Program Repair** (2016) [98] uses dynamic synthesis based on the debug interface of the JVM for repairing conditions.

• **Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis** (2016) [110] optimizes symbolic execution in order to obtain more than one angelic value, being called together called “angelic forest”, in order to synthesize multipoint patches.

• **Qlose: Program Repair with Quantitative Objectives** (2016) [97] tries to minimize the semantic impact of the repair, by minimizing the number of inputs for which there is a behavioral change using the Sketch synthesis system.


• **Automatic Error Elimination by Horizontal Code Transfer Across Multiple Applications** (2015) [108] transfers check-exit pairs between two applications to avoid crashes due to out of bounds access, integer overflow, and divide by zero errors.

• **Automatic Repair of Infinite Loops** (2015) [86] describes a patch generation system for infinite loops.

• **Relifix: Automated Repair of Software Regressions** (2015) [94] defines 8 repair templates that are specific to regression bugs.

• **Repairing Programs with Semantic Code Search** (2015) [84] repairs programs with snippets that can be semantically indexed and queried in SMT.

• **Staged Program Repair with Condition Synthesis** (2015) [88] combines condition repair à la Nopol and repair templates à la PAR.

• **DirectFix: Looking for Simple Program Repairs** (2015) [90] demonstrates that, under strong assumptions, we can state the repair problem as a Maximum Satisfiability (MaxSAT), where the smallest patch is the one that satisfies the most constraints.

• **Minthint: Automated Synthesis of Repair Hints** (2014) [64] hints to change the RHS of a single assignment statement based on data collected with concolic execution.

• **Diagnosis and Emergency Patch Generation for Integer Overflow Exploits** (2014) [75] does automatic repair of integer overflow with three repair operators: taking an error branch before the overflow happens, taking an error branch after the overflow has happened, and forced program stop.

• **Automatic Patch Generation Learned From Human-Written Patches** (2013) [49] defines 10 repair templates for fixing bugs such as (add null pointer check, etc).

• **SemFix: Program Repair via Semantic Analysis** (2013) [56] combines symbolic execution and component-based synthesis to fix boolean and integer expressions in C programs.
• **Evolving Patches for Software Repair** (2011) [30] describes pyEdb, a mutation based repair approach with two mutation operators (relational operator change and name switch) in Python.


• **Automatically Finding Patches Using Genetic Programming** (2009) [20] is the seminal paper of the field, introducing GenProg, with its sister papers *A Genetic Programming Approach to Automated Software Repair* [17], *GenProg: a Generic Method for Automatic Software Repair* [41], *Automatic Program Repair with Evolutionary Computation* [29].


1.2 Using Crashes


• **Fixing Recurring Crash Bugs via Analyzing Q&A Sites** (2016) [80] repairs exception bugs based on potential solutions found on Stackoverflow.

• **Automatic Repair of Infinite Loops** (2015) [86] repairs infinite loops with the same repair concept as Nopol.

• **CLOTHO: Saving Programs from Malformed Strings and Incorrect String Handling** (2016) [78] is a system that generates simple catch blocks to handle certain runtime exceptions related to string manipulation in Java.

For null dereferences (null pointer exceptions):


• **Dynamic Patch Generation for Null Pointer Exceptions Using Metaprogramming** (2017) [116] introduces the idea of exploring the repair search space with a meta-program and realizes it for crashing null pointer exceptions.

1.3 Using a Reference Implementation

• **Dynamic Neural Program Embedding for Program Repair** (2018): Wang et al. [128] compute an embedding on program traces in order to predict the kind of bug in student’s programs from a MOOC.

• **Automated Clustering and Program Repair for Introductory Programming Assignments** (2016): Gulwani et al.’s technique [101] modifies, inserts, and deletes statements in student’s programs while preserving the control-flow.
• Search, Align, and Repair: Data-Driven Feedback Generation for Intro-
ductive Programming Exercises (2018): Wang et al. [190] use advanced AST
matching and differencing to provide a small diff to MOOC students based on a pool
of correct solutions.

• Semantic program repair using a reference implementation (2018): Mehtaev
et al. [172] use a reference implementation and a parameterized test to generate a
patch that changes an expression with primitive values.

• Neuro-symbolic program corrector for introductory programming assign-
ments (2018): Bhatia et al. [144] combine token sequence learning and Sketch to
repair MOOC student submissions in Python. Extension of [96].

• Automatic Diagnosis and Correction of Logical Errors for Functional Pro-
gramming Assignments (2018): Lee et al. [163] present a system for automati-
cally generating feedback on logical errors in functional programming assignments in
OCaml.

• Automated Feedback Generation for Introductory Programming Assign-
ments (2013): Singh et al. [58] generate feedback for student programs based on
a reference implementation, using Sketch as an intermediate languages to search for
patches.

• Automated Error Localization and Correction for Imperative Programs
(2011): Könighofer and Bleam’s algorithm [35] fixes the the right-hand side (RHS)
of assignments by using the reference implementation as specification and driving the
synthesis with a meta-program and SMT solving. "Repair with On-the-fly Program
Analysis" is an extension of this work.

1.4 Using Contracts

The contracts can be invariants or runtime assertions, they can be manually written or
mined.

• A Metamorphic Testing Approach for Supporting Program Repair without
the Need for a Test Oracle [104] (2016) have proposed to use metamorphic relations
as repair oracle.

• Generating Fixes From Object Behavior Anomalies [15] 2009. infers an ob-
ject usage model from executions, and then generates a fix with two repair operators
(addition and removal of method calls) so that failing runs match the inferred correct
behavior.

• Automated Fixing of Programs with Contracts [28] 2010 (journal version in
2014 [76]), uses four repair templates that consist of a snippet and an empty con-
ditional expression to be synthesized, and relies on Eiffel contacts (pre-conditions,
post-conditions, invariants) to detect and provide the fix ingredients. “Code-Based
Automated Program Fixing” [38] is an extension of this work where patches don’t
have to only use argumentless boolean methods in the patch.

• Constraint-Based Program Debugging Using Data Structure Repair [37]
2011, translates runtime data structure repair à la Demsky as source code fix sug-
suggestion.

• Specification-based Program Repair Using SAT [32] 2011, uses Alloy to repairs
assignments and conditionals bugs.
1.5 Patch Generation in Production

- **Production-Driven Patch Generation** (2016) [115] proposes to use shadow applications and shadow traffic to make regression testing in production.


- **Countering Network Worms Through Automatic Patch Generation** (2005 [3]) detect buffer overflow vulnerabilities at runtime in production, then produce a source code patch that skip the execution of the overflowing statement.

2 Program Repair of Static Errors

2.1 Static Warnings


- **Static Automated Program Repair for Heap Properties** (2018) [184] repairs static warnings for potential null dereferences found by the static analysis tool Infer.

- **MemFix: static analysis-based repair of memory deallocation errors for C** (2018) [162] quantitatively improves over [81] and is able to handle real open-source programs.

- **IntPTI: Automatic Integer Error Repair With Proper-Type Inference** (2017) [114] statically detect integer overflows, applies 3 transformations (sanity check, explicit type casting and declared type change) before proposing the change to the developer.

- **Safe Memory-leak Fixing for C Programs** (2015) [81] proposes an approach that consists of statically detecting and fixing memory leaks by inserting a deallocation statement.

- **Automated Generation of Buffer Overflows Quick Fixes Using Symbolic Execution and SMT** (2015) [91] uses parametrized templates to fix buffer overflow, where the actual parameter is found with symbolic execution and SMT.

- **Sound Input Filter Generation for Integer Overflow Errors** (2014) [66] uses a static analysis specific to integer arithmetic that detects integer overflows, and repair them by inferring a filter that simply deny the input.

- **R2Fix: Automatically Generating Bug Fixes From Bug Reports** (2013) [53] takes as oracle a manually written bug report, which is used to extract the actual value of a template parameter.

- **Automatic Repair of Overflowing Expressions with Abstract Interpretation** (2013) [54] statically detects arithmetic overflow and suggest fixes as re-ordering of the arithmetic operations.
• Modular and Verified Automatic Program Repair (2012) [43] proposes a repair approach for a set of fault class identified statically (e.g., off-by-one errors), with specific repair operators per fault class (for example adding a precondition).


• A Formal Approach to Fixing Bugs [34] 2011. fixes Findbugs-like bugs with Coccinelle-like templates using a transformation language called Tran. Similar work by the same authors “Towards the Automated Correction of Bugs”.

• Automatic Error Correction of Java Programs [24] 2010. generates a meta-program that integrates all possible mutations according to a mutation operator, and the successful mutations are identified using symbolic execution.

• Using Mutation to Automatically Suggest Fixes for Faulty Programs (2010) Debroy and Wong [22] propose to use standard mutations from the mutation testing literature to fix programs: replacement of an arithmetic, relational, logical, increment/decrement, or assignment operator by another operator from the same class; decision negation in an if or while statement.

• Proof-directed Debugging and Repair (2006) [4] uses an Isabel proof-based oracle on on ML programs: when the proof fails, the counter-example of the proof drives a repair approach based on repair templates (replacing one method call by another, adding code).


2.2 Compiler Errors


• Syntax and sensibility: Using language models to detect and correct syntax errors (2018): Santos’ approach [177] repairs syntax errors (one character edits) with n-gram and LSTM, with an evaluation on 1,715,312 before-and-after pairs of the BlackBox dataset.

• Compilation error repair: for the student programs, from the student programs (2018): Ahmed et al. [139] improve over DeepFix [118] on a dataset containing a total of 16985 (source, target) line pairs.

• DeepFix: Fixing Common C Language Errors by Deep Learning (2017): Gupta et al. [118] use a language model for repairing syntactic compilation errors


3 Empirical Studies for Program Repair

• SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair (2018) [147] does sequence-to-sequence learning over 35578 diffs from the CodRep dataset [145] and shows that the system, called Sequencer, is able to perfectly predict the fixed line for 950/4711 testing cases and 14 bugs in Defects4J.
• Learning to Generate Corrective Patches using Neural Machine Translation (2019) [154] trains a neural sequence-to-sequence model over 35,137 single statement diffs from 5 open-source Java projects and applies to 233 testing tasks.

• Human-competitive Patches in Automatic Program Repair with Repairnator (2018) [174] shows that the state of the art techniques in 2018 can produce a valuable patch faster than human developers.

• Attention Please: Consider Mockito when Evaluating Newly Released Automated Program Repair Techniques (2018) [191] discusses the characteristics of the Mockito bugs in Defects4J and the performance of SimFix, CapGen and Nopol on repairing them.

• The Remarkable Role of Similarity in Redundancy-based Program Repair (2018) [146] describes an original experiment showing that the use of similarity can reduce the search space of program repair by 99.35%, under certain assumptions.

• LSRepair: Live Search of Fix Ingredients for Automated Program Repair (2018) [164] compares three kinds of similarity (similar method signature, method embedding similarity using CNN, semantic similarity based on code-search) in the context of generate-and-validate program repair.

• A Novel Fitness Function for Automated Program Repair Based on Source Code Checkpoints (2018) [180] uses instrumentation in order to have a fitness function that has less plateaus than with only test case outcomes.

• A Comprehensive Study of Automatic Program Repair on the QuixBugs Benchmark (2018) [195] is the first report on doing automatic repair on the Quixbugs benchmark, using the Astor and Nopol tools [119].

• Alleviating Patch Overfitting with Automatic Test Generation: A Study of Feasibility and Effectiveness for the Nopol Repair System (2018) [196] shows that using tests that are generated against the buggy version of the program under repair poses a serious oracle problem.

• Comparing Line and AST Granularity Level for Program Repair using PyGGI (2018) [141] claims that AST analysis in a GenProg-like approach is overall faster than line-based analysis.

• Comparing Developer-Provided to User-Provided Tests for Fault Localization and Automated Program Repair (2018) [159] studies whether the results of fault localization change if one removes the failing test case provided in the commit (experiments on Defects4J).

• An empirical analysis of the influence of fault space on search-based automated program repair (2017) [129] shows that GenProg finds more patches (incl. correct ones) if one assumes better fault localization.

• A correlation study between automated program repair and test-suite metrics (2017) [137] sets up a protocol based on held-out tests to show that the better the coverage, the better the repair.

• Do automated program repair techniques repair hard and important bugs? (2017) [123] suggests that the considered state-of-the-art repair techniques only repair simple bugs according to collected bug metadata.

• Overfitting in semantics-based automated program repair (2018) [161] compares Angelix and variants of it on the IntroClass and CodeFlaws benchmarks showing that 50-75% of patches are overfitting.
• **An Empirical Investigation into Learning Bug-Fixing Patches in the Wild via Neural Machine Translation** (2018) [186] uses machine translation on Java methods that are smaller than 50 tokens with abstracted token sequences.

• **Towards reusing hints from past fixes - An exploratory study on thousands of real samples** (2018) [198] confirms the results of [68] regarding redundancy-based repair based on the novel usage delta dependency graphs.

• **Mining Repair Model for Exception-Related Bug** (2018) [197] studies the most common repair actions per exception type for exception bug.


• **A feasibility study of using automated program repair for introductory programming assignments** (2017) [136] studies the application of GenProg, AE, Angelix, and Prophet to 661 programs written by the students taking an introductory programming course.

• **Empirical Study on Synthesis Engines for Semantics-Based Program Repair** (2016) [106] compares 5 synthesis engines implemented on top of Angelix showing that they do not have the same performance, and that Angelix's Partial MaxSMT-based synthesis engine is the best on the considered benchmark, IntroClass.

• **Sorting and Transforming Program Repair Ingredients via Deep Learning Code Similarities** (2016) [130] uses deep learning to match donor methods that are similar to the buggy method under repair.

• **Automatic Repair of Real Bugs in Java: A Large-Scale Experiment on the Defects4J Dataset** (2016) [121] is the first experiment ever on evaluating automatic repair on the Defects4J dataset (with Nopol, jGenProg and jKali) showing the great problem of overfitting.

• **Improved Crossover Operators for Genetic Programming for Program Repair** (2016) [111] proposes new crossover operators for Genprog, that decouple fix location, repair type, and repair ingredient. The corresponding journal paper is [175].

• **An Analysis of Patch Plausibility and Correctness for Generate-And-Validate Patch Generation Systems** (2015) [92] shows that most Genprog patches simply remove code and consequently that the overfitting problem is huge.

• **Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair** (2015) [93] is the first paper to name the overfitting problem.

• **The Strength of Random Search on Automated Program Repair** (2014) [71] shows that there the search in Genprog is actually not guided by the fitness function, it’s random search.

• **Automatically Generated Patches As Debugging Aids: a Human Study** (2014) [74] asks to 95 participants to fix bugs with either fault localization or machine-generated patches from PAR.

• **Do the Fix Ingredients Already Exist? An Empirical Inquiry into the Redundancy Assumptions of Program Repair Approaches** (2014) [68] shows that a significant proportion of commits in open-source projects (3%-22%) are composed of existing code.

• **Mining Software Repair Models for Reasoning on the Search Space of Automated Program Fixing** (2013) [89] computes the prevalence of each repair action and explores the imbalance between possible repair actions at the AST level, showing its significant impact on the search.
• **A Human Study of Patch Maintainability** (2012) [39] conducted a study of Genprog patches based on 150 participants and 32 real-world defects, showing that machine-generated patches are slightly less maintainable than human-written ones.

• **A Systematic Study of Automated Program Repair: Fixing 55 Out of 105 Bugs for $8 Each** (2012) [40] has famously claimed that 52% of bugs (55/105) of bugs can be fixed by Genprog, a ratio being undermined by the benchmark selection biases and by overfitting.

• **Automated Program Repair Through the Evolution of Assembly Code** (2010) [26] shows the feasibility of Genprog-like repair on binary x86 code and Java bytecode.

• **Designing Better Fitness Functions for Automated Program Repair** (2010) [23] explores the design space of fitness functions of Genprog.

4 Targeted Repair

4.1 Test Repair

• **Visual web test repair** (2018) [181] repairs broken Selenium tests by changing the incorrect locator, the locator being inferred by comparing visual renderings (ie images).

• **Waterfall: An incremental approach for repairing record-replay tests of web applications** (2016) [102] repairs DOM locators in Selenium tests.

• **Repairing Selenium Test Cases: an Industrial Case Study about Web Page Element Localization** (2013) [52] do test repair in the context of Selenium tests, which are tests for web applications with HTML output.

• **ReAssert: Suggesting Repairs for Broken Unit Tests** (2009) [16] addresses the dual problem of test-suite based repair: changing the tests instead of fixing the application.


4.2 Automated Repair of Concurrency errors

• **Understanding and Generating High Quality Patches for Concurrency bugs** (2016) [107] has proposed a tool called HFix whose repair operator is to add thread-join instructions.

• **Automatic Repair for Multi-threaded Programs with Deadlock/Livelock Using Maximum Satisfiability** (2014) [65] inserts locks by encoding the problem as a satisfiability one.

• **Axis: Automatically Fixing Atomicity Violations Through Solving Control Constraints** (2012) [42] addresses the problem of violation fixing as a constraint solving problem using the Petri net model.

• **Automated Atomicity-violation Fixing** (2011) [33] is about AFix, whose repair model consists of putting instructions into critical regions.

4.3 Automated Repair of Build Scripts

• **HireBuild: an automatic approach to history-driven repair of build scripts** (2018) [153] mines and apply build-fix patterns in Gradle, and apply them based on log similarity.
4.4 Repair for the Web

- **Fully Automated HTML and Javascript Rewriting for Constructing a Self-healing Web Proxy** (2018) [149] uses a proxy to intercept browser errors and repair them with HTML and Javascript rewriting strategies.
- **Automated repair of mobile friendly problems in web pages** (2018) [168] explores the search space of CSS modifications to fix mobile problems such as font sizing and extraneous spacing.
- **Automated Repair of Internationalization Presentation Failures in Web Pages Using Style Similarity Clustering and Search-Based Techniques** (2018) [169] fixes web rendering by changing the value of CSS properties.
- **Fix Me Up: Repairing Access-Control Bugs in Web Applications.** [59] 2013, repairs access-control policies in web applications, using a static analysis and transformations tailored to this domain.
- **Automated Repair of HTML Generation Errors in PHP Applications Using String Constraint Solving** [45] 2012, fixes incorrect opening/closing HTML tags in PHP application by encoding the problem as string constraints.

4.5 Repair of Software Models

- **Towards Automated Inconsistency Handling in Design Models** [27] (2010) uses Prolog to propose a repair plan that fixes inconsistencies in UML models.
- **Supporting Automatic Model Inconsistency Fixing** [21] (2009) detects and fixes inconsistencies in MOF and UML models.

4.6 Misc Repair Types

- **Automatic Software Merging using Automated Program Repair** (2019) [208] fixes merge conflicts with a search-based approach based on kGenProg.
- **Interactive Testing and Repairing of Regular Expressions** (2018) [142] proposes an interactive technique to repair regular expressions, the developer being asked for validation.
- **Towards Specification-Directed Program Repair** (2018) [178] does program repair for the educational programming language Karel, by training a neural net to predict the edit (keep, delete, insert or replace token).
- **A Framework for the Automatic Correction of Constraint Programs** (2011) [36] repairs constraint programs the repair consisting of declaratively removing or adding new constraints.
5 Optimization & Integration

5.1 Driving the Search

- Leveraging Program Invariants to Promote Population Diversity in Search-Based Automatic Program Repair (2019) [199] explores the usage of learned invariants to improve the fitness function of generate-and-validate program repair, experimenting with genprog4java.

- Identifying Patch Correctness in Test-Based Program Repair (2018) Xiong et al. [194] analyze test execution traces to filter out incorrect overfitting patches.

- Identifying Test-suite-overfitted Patches Through Test Case Generation (2017) Xin and Reiss [132] generate test cases with Evosuite that show behavioral differences between the patched program and the ground truth patch. (code)

- A new word embedding approach to evaluate potential fixes for automated program repair (2018) [140] computes source code line embeddings from word2vec embeddings in order to calculate distances between patches.

- History Driven Program Repair (2016) [105] uses the commit history to select the most likely patch from classical mutation-based repair (incl. Genprog and Par): the mutations that appear the most frequently in the history are ranked first.

- Prophet: Automatic Patch Generation via Learning From Successful Patches (2016) [108] selects the SPR generated patch that resembles the most to past human patches.

5.2 Improvement of the Fault Localization Step


5.3 Repair Speed

- Test-equivalence Analysis for Automatic Patch Generation [173] (2018) reduces the number of test executions in the repair loop by clustering candidate patches according to their test behaviors.

- Improving performance of automatic program repair using learned heuristics [126] 2017, uses 24 code features to identify line/expression pairs that are likely to work together, i.e. to select good candidate ingredients in redundancy based approaches.

- Leveraging program equivalence for adaptive program repair: Models and first results [60] (2013) discards some repair candidates using program equivalent checks typical from compilers.

• More Efficient Automatic Repair of Large-scale Programs Using Weak Re-compilation [44] (2012) creates an incremental compilation system that is dedicated to program repair.

5.4 Integration in Open-Source / Industry
• SapFix: Automated End-to-End Repair at Scale (2019) [205] describes the FaceBook implementation of automatic repair of null pointer exceptions found by the fuzzing tool Sapienz.
• How to Design a Program Repair Bot? Insights from the Repairnator Project (2018) [187] is the first ever blueprint architecture on using program repair in continuous integration.

6 Position Papers
• Beyond testing configurable systems: applying variational execution to automatic program repair and higher order mutation testing (2018) [193] suggests using variational execution to find multi-location repair out of a meta-program with all possible changes.
• Trusted software repair for system resiliency (2016) [112] is a 4-page position paper about detecting behavioral differences between patches using targeted differential testing.
• When App Stores Listen to the Crowd to Fight Bugs in the Wild (2015) [83] sets the vision of an App store that monitors and fixes bugs in production by orchestrating the search over thousands of devices.
• A Critical Review of "Automatic Patch Generation Learned from Human-Written Patches": Essay on the Problem Statement and the Evaluation of Automatic Software Repair (2014) [69] states that program repair goes beyond mimicking human patches, and that scientific evaluation in this research field must be designed accordingly.
• Two Flavors in Automated Software Repair: Rigid Repair and Plastic Repair (2013) [55] is an early categorization of the field, later called as generate-and-validate approaches versus semantic-based or synthesis-based approaches.
• Current Challenges in Automatic Software Repair (2013) [51] shows the vision of C. Le Goues at the end of her seminal PhD thesis on GenProg.

7 Formal Approaches to Program Repair
• Deductive Program Repair (2015) [85] does program repair for a "purely functional subset of Scala", evaluated on seeded bugs on small programs.
• Cost-Aware Automatic Program Repair (2014) [72] repairs boolean programs with assertions, by using the method of inductive assertions.
• Program Repair As Sound Optimization of Broken Programs (2009) [19] theoretically defines repair for an ad hoc formal language.
• Repair of Boolean Programs with An Application to C (2006) [5] repairs a specific class of programs called boolean programs: those that only contain boolean variables.
• **Program Repair As a Game** (2005) [2] repair programs that are expressed in linear temporal logics

8 Miscellaneous

8.1 Benchmarks

• **BugSwarm: Mining and Continuously Growing a Dataset of Reproducible Failures and Fixes** (2019) [200] uses Travis CI as [204] to collect 3,091 bugs and encapsulates them in a reproducible Docker image.

• **Bears: An Extensible Java Bug Benchmark for Automatic Program Repair Studies** (2018) Madeiral et al. [204] propose a new benchmark whose novelty is to be based on continuous integration analysis (and not on past commits).


• **Bugs.jar: a large-scale, diverse dataset of real-world Java bugs** (2018) [176] describes a dataset of 1,158 bugs and patches, over 8 open-source projects.

• **QuixBugs: a multi-lingual program repair benchmark set based on the quixey challenge** (2017) [119] is a benchmark of simple programs bugs where each bug is available in both Java and Python.

• **The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs** (2015) ManyBugs [87] is the classical GenProg benchmark and has 185 bugs in 9 C open-source programs. IntroClass is composed of small (10-20 LOC) student programs, it has been translated to Java (IntroClassJava [99]).

• **Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs** (2014) Just et al. [63] presents the Defects4J benchmark, extensively used in program repair research since the initial experiment by Durieux et al. [79, 121].

8.2 Automatic Hardening

• **Automatically Fixing C Buffer Overflows Using Program Transformations** (2014) [73] uses three program transformations dedicated to integer operations, and shows that the approach scales to real programs.

• **Program Transformations to Fix C Integers** (2013) [47] proposes three program transformations to fix common overflow problems with integer arithmetics in C code.

• **A Source-to-source Transformation Tool for Error Fixing.** (2013) [48] automatically adds a condition checks after all method calls with a source-to-source transformation in C code.


8.3 Surveys

• **Automated Program Repair** [201] (2019)

• **A Survey of Test Based Automatic Program Repair** [166] (2018)

• **Automatic software repair: a Survey** [117] (2017)

• **Automatic software repair: a Bibliography** [122] (first online, 2015, journal 2017)
8.4 Doctoral Theses

- Hua, “Unifying Program Repair and Program Synthesis”, 2018 [156]
- Tan, “Design of repair operators for automated program repair”, 2018 [182]
- Timperley, “Advanced Techniques for Search-Based Program Repair”, 2017 [127]
- Gopinath, “Systematic techniques for more effective fault localization and program repair”, 2016 [100]
- Cornu, “Automatic Analysis and Repair of Exception Bugs for Java Programs”, 2015 [77]
- Martinez, “Extraction and Analysis of Knowledge for Automatic Software Repair”, 2014 [67]

References


