The Living Review on Automated Program Repair

Martin Monperrus

The Living Review on Automated Program Repair

Martin Monperrus

version of August 9, 2022, http://bit.ly/2CehUt5

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 criteria The inclusion criteria are that the considered papers 1) must be about automatic repair with some kind of patch generation (runtime repair without patch generation is excluded[2]); 2) must be a full-length research paper (typically >10 double-column pages); 3) are stored on an durable site (notable publisher, arXiv, Zenodo). There is no restriction about whether the paper has been formally peer-reviewed or not. I will stop the living review once we reach the 500th reference.

Originality Compared to formal surveys [134, 127], this living review contains very recent references and continues to evolve. It uses a bullet-list concise style that is not typical academic writing.

Notification To get notified with new versions, click here.

Feedback Do not hesitate to report a mistake, a confusing statement or a missing paper, monperrus@kth.se.

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

```
@techreport{repair-living-review,
  title = { The Living Review on Automated Program Repair },
  author = { Martin Monperrus },
  number = { hal-01956501 },
  institution = { HAL/archives-ouvertes.fr },
  year = { 2018 }
}
```


[1] https://en.wikipedia.org/wiki/Living_review

[2] the scope of my previous survey [134] was larger, it also discussed runtime repair

Version history

- August 2022, version with 401 references, recent ones identified by ∀
- Oct 2021, version with 367 references, recent ones identified by 🕸
- Dec 2020, version with 315 references, recent ones identified by ♆
- July 2020, version with 296 references, recent ones identified by ☊
- March 2020: version with 279 references, recent ones identified by ⁜
- December 2019: version with 264 references, recent ones identified by ★
- September 2019: version with 253 references, recent ones identified by Ω
- June 2019: version with 229 references, recent ones identified by ¤
- 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

Contents

1 Program Repair of Dynamic Errors

1.1 Using Tests

- **ReFixar: Multi-version Reasoning for Automated Repair of Regression Errors** (2021, ∀) Le et al. [349] design 12 repair templates tailored to fixing regressions, evaluated on 51 regression bugs.

- **A Novel Approach For Search-Based Program Repair** (2021, ⚝) Trujillo et al. [371] create a variant of GenProg integrating Lehman and Stanley's 'novelty search' to promote exploration and diversity of patches.

- **VarFix: Balancing Edit Expressiveness and Search Effectiveness in Automated Program Repair** (2021, ⚝) Wong et al. [372] combine GenProg single edits into a metaprogram to identify those combinations that pass all tests.

- **FlexiRepair: Transparent Program Repair with Generic Patches** (2020, ♈), Koyuncu et al. [306] present a repair pipeline built on top of the Coccinelle engine for semantic patches.

- **Astor: Exploring the Design Space of Generate-and-Validate Program Repair beyond GenProg** (2019, ⍾) Martinez et al. [256] identify 12 dimensions in the design space of generate-and-validate program repair and implement them as extension points in the Astor framework.

- **Impact Analysis of Syntactic and Semantic Similarities on Patch Prioritization in Automated Program Repair** (2019, ★) Asad et al. [220] propose an alternative patch ranking technique for CapGen.

- **SOSRepair: Expressive Semantic Search for Real-World Program Repair** (2019, ★) Afzal et al. [219] proposes a better encoding than [86] to repair C programs with SMT-based snippet search.

- **LoopFix: An Approach to Automatic Repair of Buggy Loops** (2019,Ω) Wang et al. [272] describe a system that changes either the loop condition or an assignment in the loop body, using symbolic execution and component-based synthesis.

- **Automatic patch generation with context-based change application** (2019,Ω) Kim and Kim [239] present ConFix, that first searches for past patches with surrounding code similar to the suspicious code locations (based on a hash of the AST) and when a context matches, the past change is ported to the suspicious location.

- **Harnessing evolution for multi-hunk program repair** (2019,Ω) Saha et al. [264] mine repair locations that evolve together in order to search for patches consisting on the same systematic edit done at different locations.

- **TBar: Revisiting Template-based Automated Program Repair** (2019,¤) Liu et al. [246] consolidate 35 fix patterns in 15 categories and measure their effectiveness over Defects4J.

- **Ultra-Large Repair Search Space with Automatically Mined Templates: the Cardumen Mode of Astor** (2018) [189] shows that parametrized repair ingredients yields an explosion of the repair search space and finds 8935 Patches for Defects4J.

- **Mining Stackoverflow for Program Repair** (2018) Liu and Zhong [184] clusters AST diffs from code pairs in Stackoverflow to extract 12 repair patterns.

- **Towards practical program repair with on-demand candidate generation** (2018) [175] does repair with metaprograming as [126] in order to explore the search space of variable and literal replacement.

- **CFAAR: Control Flow Alteration to Assist Repair** (2018) [204] uses specific patterns to determine angelic values à la Nopol [149] (eg switch only the first execution of the condition).

- **Context-Aware Patch Generation for Better Automated Program Repair** (2018) [212] 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) [167] revisits Schulte's work [27] for Java bytecode and Defects4J.

- **Program Repair via Direct State Manipulation** (2018) [173] 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) [136] creates a meta-program parametrized with parameters, encoding the search space: the symbolic solution to satisfy all test constraints is the patch. The tool is called CETI.

- **Contract-based Program Repair Without the Contracts** (2017) Chen et al. [122] uses 5 repair templates, called schemas, with a focus on modifying the state by adding an assignment. (code, journal version: [290])

- **Precise Condition Synthesis for Program Repair** (2017) Xiong et al. [148] 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 [147] use Tf-Idf similariy 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 [274, 273].

- **ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming** (2017) [155] combines 3 different techniques (patch representation, multi-objective search, method scope) to improve a GenProg-based repair loop. ARJA-e [280, 324] is an improvement over Arja integrating templates and repair anti-patterns.

- **ELIXIR: Effective Object Oriented Program Repair** (2017) [137] proposes 8 repair patterns à la PAR [51] to be used together with simple enumeration-based synthesis.

- **ASTOR: A Program Repair Library for Java** (2016) [116] presents the Java framework in which jGenProg [133], jKali [133], DeepRepair [146], Cardumen [189] are implemented.

- **Automated Program Repair by Using Similar Codfe Containing Fix Ingredients** (2016) [108] modifies RSRepair [73] in order to select the most similar repair ingredients first.

- **DynaMoth: Dynamic Code Synthesis for Automatic Program Repair** (2016) [103] uses dynamic synthesis based on the debug interface of the JVM for repairing conditions.

- **Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis** (2016) [117] 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) [102] 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.

- **Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs** (2016) [149] addresses two classes of bugs: buggy if conditions and missing preconditions. Initial paper: "Automatic Repair of Buggy If Conditions and Missing Preconditions with SMT" [63].

- **Automatic Repair of Infinite Loops** (2015) [88] describes a patch generation system for infinite loops.

- **Relifix: Automated Repair of Software Regressions** (2015) [99] defines 8 repair templates that are specific to regression bugs.

- **Repairing Programs with Semantic Code Search** (2015) [86] repairs programs with snippets that can be semantically indexed and queried in SMT.

- **Staged Program Repair with Condition Synthesis** (2015) [91] combines condition repair à la Nopol and repair templates à la PAR.

- **DirectFix: Looking for Simple Program Repairs** (2015) [93] 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) [66] 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) [77] 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) [51] defines 10 repair templates for fixing bugs such as (add null pointer check, etc).

- **SemFix: Program Repair via Semantic Analysis** (2013) [58] combines symbolic execution and component-based synthesis to fix boolean and integer expressions in C programs.

- **Evolving Patches for Software Repair** (2011) [31] describes pyEdb, a mutation based repair approach with two mutation operators (relational operator change and name switch) in Python.

- **On the Automation of Fixing Software Bugs** (2008) [11] defines 7 mutation operators based on abstract syntax tree modification in a prototype implementation called Jaff, that handles a subset of Java. Journal version is "Evolutionary Repair of Faulty Software" [32]. Another version is "A Novel Co-evolutionary Approach to Automatic Software Bug Fixing" [12].

- **Automatically Finding Patches Using Genetic Programming** (2009) [21] is the seminal paper of the field, introducing GenProg, with its sister papers **A Genetic Programming Approach to Automated Software Repair** [18], **GenProg: a Generic Method for Automatic Software Repair** [42], **Automatic Program Repair with Evolutionary Computation** [30].

- **BugFix: a Learning-based Tool to Assist Developers in Fixing Bugs** (2009) [19] suggests a bug fix action using association rules based on features on the suspicious statement.

1.2 Using Crashes

- **Exception-Driven Fault Localization for Automated Program Repair** (2022, ∀) Ginelli et al. [385] describe a template based repair technique where templates are associated to specific Java exceptions.

- **Beyond Tests: Program Vulnerability Repair via Crash Constraint Extraction Gao** (2021, 🕸) Gao et al. [341] use sanitizers to obtain clean crashes and fix conditional expressions (if, loops) to avoid the crash. The prototype tool is called ExtractFix, and is available as Docker image on gaoxiang9430/extractfix.

- **Crash-avoiding program repair** (2019,¤) Gao et al. [230] repair crashes in C code with three operators (assigments, if-condition, precondition) using implicit oracles and fuzzing to discard incorrect patches.

- **Repairing crashes in Android apps** (2018) [202] defines 8 repair operators tailored for Android crashes.

- **Production-Driven Patch Generation** (2016) [125] proposes to use shadow applications and shadow traffic to make regression testing in production.

- **Fixing Recurring Crash Bugs via Analyzing Q&A Sites** (2016) [82] repairs exception bugs based on potential solutions found on Stackoverflow.

- **Automatic Repair of Infinite Loops** (2015) [88] repairs infinite loops with the same repair concept as Nopol.

- **CLOTHO: Saving Programs from Malformed Strings and Incorrect String Handling** (2016) [80] is a system that generates simple catch blocks to handle certain runtime exceptions related to string manipulation in Java.

- **Automatic Error Elimination by Horizontal Code Transfer Across Multiple Applications** (2015) [114] transfers check-exit pairs between two applications to avoid crashes due to out of bounds access, integer overflow, and divide by zero errors.

For null dereferences (null pointer exceptions):

- **NPEX: Repairing Java Null Pointer Exceptions without Tests** (2022, ∀) Lee et al. [389] devise a bespoke symbolic execution technique to avoid incorrect patches when repairing null pointer exceptions in Java without tests. The system is evaluated on 119 NPEs and available on Github.

- **VFix: Value-Flow-Guided Precise Program Repair for Null Pointer Dereferences** (2019): VFix [276] ranks patches for null pointers based on congested places: those places in the data-flow graph that maximize the likelihood of fixing many NPEs at once.

- **Automatic Inference of Code Transforms for Patch Generation** (2017): Long et al. [131] infers repair schemas from past commits for Java's NullPointerException and OutOfBoundsException.

- **Dynamic Patch Generation for Null Pointer Exceptions Using Metaprogramming** (2017) [126] 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 / Feedback Generation

In this section, many papers are in the context of automated feedback generation for students, where a reference solution to a programming exercise exists.

- **FAPR: Fast and Accurate Program Repair for Introductory Programming Courses** (2021, 🕸) Lu et al.'s technique [352] consists of generating a meaningful high level feedback based on a low-level token edit script.

- **Re-factoring based Program Repair applied to Programming Assignments** (2019, ★) [235] is a feedback generation technique based on the idea of generating equivalent refactored programs so as to find a correct program which has the same control flow structure as the buggy student Python program under consideration.

- **Dynamic Neural Program Embedding for Program Repair** (2018): Wang et al. [144] 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 [106] modifies, inserts, and deletes statements in student's programs while preserving the control-flow.

- **Semantic program repair using a reference implementation** (2018): Mechtaev et al. [191] 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 assignments** (2018): Bhatia et al. [161] combinetoken sequence learning and Sketch to repair MOOC student submissions in Python. Extension of [101].

- **Automatic Diagnosis and Correction of Logical Errors for Functional Programming Assignments** (2018): Lee et al. [181] present a system for automatically generating feedback on logical errors in functional programming assignments in OCaml.

- **Automated Feedback Generation for Introductory Programming Assignments** (2013): Singh et al. [60] 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 [36] 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.

- **Input Test Suites for Program Repair: A Novel Construction Method Based on Metamorphic Relations** (2020, Ɏ) Jiang et al. [303] define metamorphic relations for the Siemens benchmark and execute Angelix, CETI, and GenProg to fix the Siemens faults accordingly.

- **Program Repair at Arbitrary Fault Depth** (2019,¤) Khaireddine et al. [238] modifies the patch validation step of Astor/jGenProg [257] to use an absolute correctness formula and a strict relative correctness relation.

- **A Metamorphic Testing Approach for Supporting Program Repair without the Need for a Test Oracle** (2016) Jiang et al. [109] have proposed to use metamorphic relations as repair oracle.

- **Generating Fixes From Object Behavior Anomalies** (2009) [16] Dallmeier et al. infer an object 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** (2010, journal version in 2014 [78]) [29], uses four repair templates that consist of a snippet and an empty conditional 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" [39] 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** (2011) [38] translates runtime data structure repair à la Demsky as source code fix suggestion.

- **Specification-based Program Repair Using SAT** (2011) [33] uses Alloy to repairs assignments and conditionals bugs.

1.5 Data-driven repair approaches

1.5.1 Data-driven Patch Generation

- **Defect Identification, Categorization, and Repair: Better Together** (2022, ∀) Ni et al. [390] train the a single system 1) to classify lines among one of 16 defect patterns and 2) to generate the fix with a decoder, experimenting on ManySStuBs4J.

- **GLAD: Neural Predicate Synthesis to Repair Omission Faults** (2022, ∀) Kang and Yoo [387] train a GRU-based system to generate if conditions at certain locations in order to early-return, guard existing code or add clauses to existing conditions.

- **Fix Bugs with Transformer through a Neural-Symbolic Edit Grammar** (2022, ∀) Hu et al.'s experiments on CodeXBlue [386] indicate that predicting the edit sequence according to an edit grammar is more effective than predicting the whole fixed code, confirming [335].

- **M3V: Multi-modal Multi-view Context Embedding for Repair Operator Prediction** (2022, ∀) Xu et al. [396] devise a graph-based neural approach to predict one repair operator among 4 standard ones for NullPointerException and 3 for OutOfBoundsException.

- **Can We Automatically Fix Bugs by Learning Edit Operations** (2022, ∀) Connor et al. [383] present a series of negative experimental results on using edit operations as output to neural program repair.

- **GrasP: Graph-to-Sequence Learning for Automated Program Repair** (2021, ∀) Tang et al. [368] design a graph based representation for generating Java patches with a graph-to-sequence neural architecture from IBM (IBM/Graph2Seq).

- **A Controlled Experiment of Different Code Representations for Learning-Based Bug Repair** (2021,) Namavar et al. [354] compare the ability of 10 token-based representations and 4 AST based representations for repairing swapped arguments, wrong binary operator, and wrong binary operands, showing a relative advantage for token-based pre-order pretty-print of original code (AST4).

- **A Syntax-Guided Edit Decoder for Neural Program Repair** (2021,) Zhu et al. [380] propose a decoder architecture for neural program repair that 1) generates edits (and not full sequences) 2) generates placeholders for handling rare identifiers (instead of subtokenization or copy [224]).

- **Grammar-Based Patches Generation for Automated Program Repair** (2021,) Tang et al. [369] proposes a neural architecture combining a token encoder and a grammar encoder, and experiment with the code changes of Tufano's BFP dataset [205].

- **CURE: Code-Aware Neural Machine Translation for Automatic Program Repair** (2021,) Jiang et al. [344] propose a subword tokenization technique and a specific beam search to improve the compilation rate of patches from NMT-based repair.

- **A Software-Repair Robot Based on Continual Learning** (2021,) Baudry et al [329] uses continual learning on top of the stream of continuous integration builds, refining the patch generation ML model when new builds arrive.

- **Synthesize, Execute and Debug: Learning to Repair for Neural Program** (2020,) Gupta et al. [299] embed execution traces in order for a so-called neural debugger to predict an edit sequence to repair Karel programs.

- **DLFix: Context-based Code Transformation Learning for Automated Program Repair** (2020,) Li et al. [307] use tree-based recurrent neural networks to generate patches.

- **CoCoNuT: Combining Context-Aware Neural Translation Models using Ensemble for Program Repair** (2020,) Lutellier et al. [252, 310] propose a number of design changes to SequenceR [224] (fully convolutional layers, multi-attention, multi-model prediction).

- **Hoppity: Learning Graph Transformations to Detect and Fix Bugs in Programs** (2020,) Dinella et ak. [293] predict the changes to be made to the AST of Javascript bug-fix commits with a graph-based neural network.

- **A Study of Pyramid Structure for Code Correction** (2020,) Huang et al. .[302] propose a better encoder for seq2seq and apply it to two benchmarks of programs with static warnings: Juliet and Java SARD.

- **Learning the Relation between Code Features and Code Transforms with Structured Prediction** (2019,) Yu et al. [279] predict the code transformations that must be applied to fix a bug using structured prediction with conditional random fields.

- **SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair** (2018) Chen et al. [224] deploy sequence-to-sequence learning over 35578 diffs from the CodRep dataset [162] and show 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 Repair Software Vulnerabilities with Generative Adversarial Networks** (2018, ★) [170] generates noisy data by removing source code tokens, this data being used to train a sequence to sequence model.

- **Learning to Generate Corrective Patches using Neural Machine Translation** (2019) [172] trains a neural sequence-to-sequence model over 35,137 single statement diffs from 5 open-source Java projects and applies it to 233 testing tasks.

- **Search, Align, and Repair: Data-Driven Feedback Generation for Introductory Programming Exercises** (2018): Wang et al. [210] use advanced AST matching and differencing to provide a small diff to MOOC students based on a pool of correct solutions.

- **Semantic Code Repair using Neuro-Symbolic Transformation Networks** (2017) Delvin et al. [124] synthesize errors in Python programs according to 4 mutation operators and show that an LSTM-based architecture can fix the synthetic errors.

- **History Driven Program Repair** (2016) [110]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) [114] selects the SPR generated patch that resembles the most to past human patches.

- **sk_p: a neural program corrector for MOOCs** (2016) Pu et al. [119] use a recurrent neural network to predict corrections in small student programs written in Pyton.

1.5.2 Inference of Fix Patterns / Templates

- **Expanding Fix Patterns to Enable Automatic Program Repair** (2021, ∀) Nowack et al. [360] cluster Defects4J patches to group them by fix pattern.

- **Type error feedback via analytic program repair** (2020,) Sakkas et al. [315] infer fix templates in OCaml for repairing type system errors in programs from students in an introductory programming course.

- **DevReplay: Automatic Repair with Editable Fix Pattern** (2020,) Ueda et aL; [317] abstracts over commits by extracting matching and replacement regular expressions, in order to be able to apply the same code change again later.

- **FixMiner: Mining Relevant Fix Patterns for Automated Program Repair** (2020,) Koyuncu et al. [305] define a novel data structure for representing and clustering edit scripts, finding 14 full patterns automatically in a dataset of 11,416 patches.

- **Phoenix: Automated Data-driven Synthesis of Repairs for Static Analysis Violations** (2019,Ω) Bavishi et al. [221] represent warning-fixing changes in a DSL representing the AST edit script, then cluster those changes into patterns.

- **Getafix: Learning to Fix Bugs Automatically** (2019) [265] infers repair templates for null pointer bugs detected with the static analysis tool Infer.

- **Shaping Program Repair Space with Existing Patches and Similar Code** (2018) [176] selects the most similar repair ingredients that are also instances of bug fix patterns mined over past commits.

2 Program Repair of Static Errors

2.1 Static Warnings

- **TFix: Learning to Fix Coding Errors with a Text-to-Text Transformer** (2021,) Berabi et al. [331] train and evaluate a T5 transformer to repair ESLint errors in Javascript.

- **Sorald: Automatic Patch Suggestions for SonarQube Static Analysis Violations** (2021,) Etemadi et al. [339] present a system to repair SonarJava static analysis warnings based on AST level metaprogramming with Spoon [96].

- **Automatic Integer Error Repair by Proper-Type Inference** (2021,) Cheng et al. [225] write a static analysis for C integer errors based on type inference, and use four fix patterns to repair the violations.

- **Automated Code Repair to Ensure Spatial Memory Safety** (2021,) Klieber et al. [347] add checks to repair warning by the verification tool Symbiotic, using an ad hoc intermediate representation that can be transformed from and back to the AST.

- **C-3PR: A Bot for Fixing Static Analysis Violations via Pull Requests** (2020, ⁜) C-3PR [288] integrates ESLint, TSLint and Sonar-WalkMod into a bot that makes pull-requests on Github for style issues and static analysis warnings.
- **SAVER: Scalable, Precise, and Safe Memory-Error Repair** (2020, ⁜) Hong et al. [301] propose a novel technique to patch statically found memory leak, double-free, and use-after-free errors in C programs based on so-called object flow graphs.
- **Automated Repair of Resource Leaks in Android Applications** (2020, ⁜) Bhatt et al. [285] repair Android-specific static analysis warnings with a fix template.
- **IntRepair: Informed Repairing of Integer Overflows** (2019, ⍾) Muntean et al. [263] use 4 repair patterns to statically repair integer overflows found with static analysis.
- **Automatically Generating Fix Suggestions in Response to Static Code Analysis Warnings** (2019,Ω) Marcilio et al. [254] fix 11 Sonarqube warnings with fixing rules implemented in the Rascal metaprogramming system.
- **Avatar: Fixing Semantic Bugs with Fix Patterns of Static Analysis Violations** (2019) Liu et al. [245] fixe 7 FindBugs warnings with carefully selected fix patterns.
- **Neural Program Repair by Jointly Learning to Localize and Repair** (2019) Vasic et al.'s [208] does joint detection and repair of variable-misuse bugs instead of Allamanis et al's technique of detection followed by enumeration.
- **Static Automated Program Repair for Heap Properties** (2018) [203] 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) [180] quantitatively improves over [83] and is able to handle real open-source programs.
- **Automatically Diagnosing and Repairing Error Handling Bugs in C** (2017) Tian et al. [141] repair three static warnings related to error handling with the corresponding template ("Incorrect/Missing Error Propagation", "Incorrect/Missing Error Checks", "Incorrect/Missing Resource Release")
- **IntPTI: Automatic Integer Error Repair With Proper-Type Inference** (2017) [123] statically detect integer overflows, applies 3 transformations (sanity check, explicit type casting and declared type change) before proposing the change to the developer.
- **Sound and complete mutation-based program repair** (2016,¤) [120] Rothenberg and Grumberg apply standard mutation operators not to the program under repair but to a constraint-based, SSA representation of C programs in order to fix statically detected errors. repo
- **Enhancing automated program repair with deductive verification** (2016, ⁜) Le at al. [112] repair static warnings found with HIP/SLEEK with Genprog-like mutations.
- **Safe Memory-leak Fixing for C Programs** (2015) [83] 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) [94] 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) [68] uses a static analysis specific to integer arithmetic that detects integer overflows, and repair them by inferring a filter that simply deny the input.

- **Automatic Repair of Overflowing Expressions with Abstract Interpretation** (2013) [56] statically detects arithmetic overflow and suggest fixes as re-ordering of the arithmetic operations

- **Modular and Verified Automatic Program Repair** (2012) [44] proposes a repair approach for a set of fault class identified statically (e.g. off-by-one errors),with a specific repair operators per fault class (for example adding a precondition).

- **Fix-it: An Extensible Code Auto-Fix Component in Review Bot** [48] (2013) is an approach to automatically fix static warnings with AST transformation based on XQuery (US Patent by the same author US9146712B2).

- **Combining dynamic slicing and mutation operators for ESL correction** (2012,¤) Repinski et al. [46] revisit the work of [23] with different mutation operators.

- **A Formal Approach to Fixing Bugs** (2011) [35] 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** (2010) [25] 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 [23] 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) [5] 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).

- **Patches As Better Bug Reports** (2006) Weimer [8] uses a safety policy of the form of a typestate property to detect and repair the control-flow graph of a method with a patch.

2.2 Bug reports

- **iFixR: bug report driven program repair** (2019,Ω) Koyuncu et al. [241] show that bug reports can be used for fault localization using information retrieval techniques and combine this with template based repair.

- **R2Fix: Automatically Generating Bug Fixes From Bug Reports** (2013) [55] takes as oracle a manually written bug report, which is used to extract the actual value of a template parameter.

2.3 Compiler Errors - Syntax Errors

- **Break-It-Fix-It: Unsupervised Learning for Program Repair** (2021, 🕸) Yasunaga and Liang [375] present a self-supervised training loop based on exercising and improving a 'breaker' and a 'fixer' simultaneously, inspired by backtranslation, in order to fix syntax errors in Python and C.

- **Self-Supervised Bug Detection and Repair** (2021, 🕸) Allamanis et al. [327] devise a self-supervised loop to detect and repair four kinds of bugs (″Variable Misuse″, ″Argument Swapping″, ″Wrong operator″, ″Wrong literal″), with experiments in Python.

- **SYNFIX: Automatically Fixing Syntax Errors using Compiler Diagnostics** (2021, 🕸) Ahmed et al.′s system [326], Synfix, uses a Roberta-based model to fix syntax errors in Java.

- **GGF: A Graph-based Method for Programming Language Syntax Error Correction** (2020, ☊) Wu et al. [319] uses the AST information in a neural architecture to improve the state-of-the-art on the DeepFix dataset.

- **Graph-based Self-Supervised Program Repair from Diagnostic Feedback** (2020, ☊) Yasunaga and Liang [322] generate training data for compiler error repair, with a self-supervised procedure based on corrupting programs, claim to improve the state-the-art on the Deepfix dataset.

- **Automatic Repair and Type Binding of Undeclared Variables using Neural Networks** (2019,Ω) Mohan et al. [260] train a system based on LSTM to repair 1059 student C programs with undeclared variable errors.

- **DeepDelta Learning to Repair Compilation Errors** (2019,Ω) Mesbah et al. [258] fix Java compilation errors by training a NMT model to predict the AST diff expressed in a textual manner.

- **SampleFix: Learning to Correct Programs by Sampling Diverse Fixes** (2019,Ω) Hajipour et al. [233] repair syntax errors with a conditional variational autoencoder with a technique to sample diverse solutions.

- **Deep Reinforcement Learning for Syntactic Error Repair in Student Programs** (2018) [169] uses reinforcement learning to improve the performance of DeepFix [128] on the same dataset.

- **Reducing Cascading Parsing Errors Through Fast Error Recovery** (2018,¤) [164] Diekmann and Tratt finds repair sequences for syntax errors, with minimum cost and acceptable time, by extending [1].

- **Syntax and sensibility: Using language models to detect and correct syntax errors** (2018): Santos′ approach [196] 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. [156] improve over DeepFix [128] 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. [128] use a language model for repairing syntactic compilation errors

- **Automated correction for syntax errors in programming assignments using recurrent neural networks** (2016): Bhatia [101] set up recurrent neural networks to fix Python syntax errors in 14000 student submissions from a MOOC.

3 Empirical Studies for Program Repair

- **A Comparative Study of Automatic Program Repair Techniques for Security Vulnerabilities** (2022, ∀) Pinconschi et al. [361] compare 10 program repair tools for C on the DARPA Cyber Grand Challenge benchmark of 250 vulnerabilities in C/C++ showing that AE and GenProg clearly yield more patches.

- **Estimating the Potential of Program Repair Search Spaces with Commit Analysis** (2022, ∀) Etemadi et al. [384] estimate the applicability of program repair by measuring the proportion of real-world commits that lie in known repair search spaces.
- **Where were the repair ingredients for Defects4j bugs?** (2021, 🕸) Yang et al. [374] study the origin of repair ingredients for redundancy-based repair and suggest that some repair ingredients may be found in test case code.
- **Evaluating Automatic Program Repair Capabilities to Repair API Misuses** (2020, 🕸) Kechagia et al. [345] compare 14 Java test-suite-based repair tools on 101 API misusage bugs. The repair tools generate patches for 28% of API misuses, 25% of the generated patches are semantically correct, TBAR has the highest number of plausible and correct patches.
- **A Comprehensive Study of Code-removal Patches in Automated Program Repair** (2020, 🕸) Ginelli et al. [298] studies code-removal patches by Astor/jKali and finds that their presence clearly indicates test weaknesses.
- **On the Impact of Flaky Tests in Automated Program Repair** (2021, 🕸) Qin et al. [363] identify environment-dependent tests in Defects4J and show that their presence impact repair results.
- **Understanding the Non-Repairability Factors of Automated Program Repair Techniques** (2020, ♈) Lin et al. [308] study the experimental logs shared in open science replication packages from program repair research, and find that the research prototypes suffer from important limitations.
- **Longitudinal Analysis of the Applicability of Program Repair on Past Commits** (2020, ♈) Etemadi et al. [296] use AST analysis to identify past commits that could potentially have been generated by program repair tools, because the corresponding code changes lie in the search space of known repair approaches.
- **Patching as Translation: the Data and the Metaphor** (2020, ♈) Ding et al. [294] discuss to what extent the usage of neural machine translation is appropriate for program repair.
- **Quality of Automated Program Repair on Real-World Defects** (2020, ♈) Motwani et al. [311] implement the algorithms of GenProg, Par and TrpAutoRepair for Java in a tool called JarFly, and study its effectiveness on Defects4J.
- **Empirical Analysis of 1-edit Degree Patches in Syntax-Based Automatic Program Repair** (2020, ☂) Dziurzanski et al. [295] exhaustively explore the search space on 1-edit patches (i.e. one-liners) of Arja for Defects4J, and show that much fewer tests can be executed for one-liners.
- **How Effective is Automated Program Repair for Industrial Software** (2020, ⋕) Noda et al. [313] discusses the repair results (8 patches) of proprietary repair tool Elixir on 20 single-statements bugs from Fujitsu products.
- **On the Efficiency of Test Suite based Program Repair** (2020, ⋕) Liu et al. [309] show that incorrect fault-localization significantly increases the chances of producing overfitting patches.
- **A manual inspection of Defects4J bugs and its implications for automatic program repair** (2019, ★) Jiang et al. [237] classify 50 Defects4J bugs with respect to the fault localization and repair stragegy used.
- **Repairnator patches programs automatically** (2019,Ω) Monperrus et al. [262] report that program repair can be human-competitive: 5 generated patches have been synthesized faster than the human developer, and accepted and merged in the code base.

- **The effectiveness of context-based change application on automatic program repair** (2019,Ω) Kim et al. [240] show that it is valuable to select ingredients with similar AST context in generate-and-validate program repair. Idea related to [212].

- **How Different Is It Between Machine-Generated and Developer-Provided Patches** (2019,¤) [271] Wang et al. asked 27 undergraduate students whether APR patches for Defects4J are correct, are located at the same position and consist of the same modification kind (132/177 patches are at the same location, with the same modification).

- **Empirical Review of Java Program Repair Tools: A Large-Scale Experiment on 2,141 Bugs and 23,551 Repair Attempts** (2019,¤) Durieux et al. [229] run the same set of repair tools over different benchmarks and show that research is likely overfitting to Defects4J.

- **Human-competitive Patches in Automatic Program Repair with Repairnator** (2018) [193] 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) [211] 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) [163] 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) [183] 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) [199] 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) [215] is the first report on doing automatic repair on the Quixbugs benchmark, using the Astor and Nopol tools [130].

- **Comparing Line and AST Granularity Level for Program Repair using PyGGI** (2018) [158] 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) [177] studies whether the results of fault localization change if one removes the failing test case provided in the commit (experiments on Defects4J).

- **The Impacts of Techniques, Programs and Tests on Automated Program Repair: An Empirical Study** (2017) Kong et al. [129] compare GenProg, RSRepair, AE and Kali on the Siemens benchmark.

- **Better test cases for better automated program repair** (2017) Yang et al. [151] use fuzz testing to generate new test cases, and employ implicit oracles (absence of crash and memory-safety) to enhance validity checking of automatically-generated patches in C.

- **An empirical analysis of the influence of fault space on search-based automated program repair** (2017) [145] 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) [153] 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) [135] suggests that the considered state-of-the-art repair techniques only repair simple bugs according to collected bug metadata.
- **An Empirical Investigation into Learning Bug-Fixing Patches in the Wild via Neural Machine Translation** (2018) Tufano et al. [205] use machine translation on Java methods that are smaller than 50 tokens with abstracted token sequences (the corresponding journal paper is [206]).
- **Towards reusing hints from past fixes - An exploratory study on thousands of real samples** (2018) [218] confirms the results of [70] regarding redundancy-based repair based on the novel usage delta dependency graphs.
- **Mining Repair Model for Exception-Related Bug** (2018) [217] studies the most common repair actions per exception type.
- **Common Statement Kind Changes to Inform Automatic Program Repair** (2018) Soto et al. [198] replicates the study of [92] on the MSR Challenge dataset.
- **A feasibility study of using automated program repair for introductory programming assignments** (2017) [152] 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) [111] 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) [146] 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) [133] 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) [118] proposes new crossover operators for Genprog, that decouple fix location, repair type, and repair ingredient. The corresponding journal paper is [194].
- **An Analysis of Patch Plausibility and Correctness for Generate-And-Validate Patch Generation Systems** (2015) [97] shows that most Genprog patches simply remove code and consequently that the overfitting problem is huge.
- **The Strength of Random Search on Automated Program Repair** (2014) [73] shows that there the search in Genprog is actually not guided by the fitness function, it's random search.
- **Do the Fix Ingredients Already Exist? An Empirical Inquiry into the Redundancy Assumptions of Program Repair Approaches** (2014) [70] 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) [92] computes the prevalence of each repair actiona dn explores the imbalance between possible repair actions at the AST level, showing its significant impact on the search.

- **A Systematic Study of Automated Program Repair: Fixing 55 Out of 105 Bugs for $8 Each** (2012) [41] 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) [27] shows the feasibilty of Genprog-like repair on binary x86 code and Java bytecode.

- **Designing Better Fitness Functions for Automated Program Repair** (2010) [24] explores the design space of fitness functions of Genprog.

3.1 Human Study on APR

- **Let's Talk With Developers, Not About Developers: A Review of Automatic Program Repair Research** (2022, ∀) Winter et al. [395] analyze published APR papers wrt to human factors and advocate for more APR research involving developers.

- **Trust Enhancement Issues in Program Repair** (2022, ∀) Noller et al. [391] collect qualitative feedback about APR from 103 developers, suggesting that developers are willing to provide additional inputs in order to increase trust in automatically generated patches.

- **Program Repair: Automated vs. Manual** (2022, ∀) Zhang et al. [399] ask 20 graduate students to repair 8 Defects4J bugs and discuss the results, suggesting that incorrect patches may be misleading for humans.

- **How to trust auto-generated code patches? A developer survey and empirical assessment of existing program repair tools** (2021, 🕸) Noller et al. [359] ask 35 questions to 100 developers about APR and suggest that trust in APR patches would increase by presenting additional artifacts (in particular generated test cases).

- **Would You Fix This Code for Me? Effects of Repair Source and Commenting on Trust in Code Repair** (2020,) Alarcon et al. [281] asked 51 programmers about their opinion on 5 GenProg patches on ManyBugs where the controlled variable is the identity of the patch author (Bill vs GenProg): the subbjects trust human-being Bill more than bot GenProg.

- **Trust in Automated Software Repair** (2019,Ω) Tyler et al. [269] ask 24 students and 24 professionals to assess 5 GenProg patches and show novice programmers are more accepting generating.

- **Characterizing Developer Use of Automatically Generated Patches** (2019,Ω) Cambronero et al. [222] performs a user study consisting of giving 5 patches on 2 bugs to 12 developers, incl. one being correct to see how developers leverage generated patches.

- **Automatically Generated Patches As Debugging Aids: a Human Study** (2014) [76] asks to 95 participants to fix bugs with either fault localization or machine-generated patches from PAR.

- **A Human Study of Patch Maintainability** (2012) [40] 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.

4 Domain-Specific Repair

4.1 Test Repair

- **GUI-Guided Test Script Repair for Mobile Apps** (2020, Ψ) Pan et al. [314] repair Android GUI test scripts by changing test UI locators or UI events, based on image and OCR analysis of GUI screenshots.

- **iFixFlakies: A Framework for Automatically Fixing Order-Dependent Flaky Tests** (2019,Ω) Shi et al. [266] analyze and repair the test bugs related to test execution ordering.

- **Intent-Preserving Test Repair** (2019,¤) Li et al. [244] repair Java tests that do not compile after evolution by ranking the candidate solutions according to an intent similarity score computed from path conditions.

- **Visual web test repair** (2018) [200] 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) [107] repairs DOM locators in Selenium tests.

- **Repairing Selenium Test Cases: an Industrial Case Study about Web Page Element Localization** (2013) [54] 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) [17] addresses the dual problem of test-suite based repair: changing the tests instead of fixing the application.

- **Automatically Repairing Event Sequence-based GUI Test Suites for Regression Testing** (2008) [13] does test repair on GUI test models. "SITAR: GUI Test Script Repair" [84] extends this work by considering manually scripted test cases.

4.2 Automated Repair of Concurrency errors

- **Automatic Detection, Validation and Repair of Race Conditions in Interrupt-Driven Embedded Software** (2022, ∀) Yu et al. [323] suggest strategies 'Add locks' (AL) or 'Interrupt disable and enable (IDE)' after a combination of static analysis and symbolic execution in oder to repair race condition problems related to hardware interrupts.

- **HIPPODROME: Data Race Repair using Static Analysis Summaries** (2021, 🕸) Hippodrome [337] repairs data races identified by RacerD, Facebook's static concurrency analyser for Java, by changing mutexes of Java synchronized blocks.

- **HangFix: automatically fixing software hang bugs for production cloud systems** (2020, Ψ) He et al. [300] propose four automatic patching strategies that are specific to software hang bugs in cloud systems such as Hadoop.

- **DFix: automatically fixing timing bugs in distributed systems** (2019,¤) Li et al. [243] fix atomicity violations, order violations, and fault-timing bugs with rollbacking side-effect operations.

- **Understanding and Generating High Quality Patches for Concurrency bugs** (2016) Liu et al. [113] have 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) Lin et al. [67] insert locks by encoding the problem as a satisfiability one.

- **Axis: Automatically Fixing Atomicity Violations Through Solving Control Constraints** (2012) [43] addresses the problem of violation fixing as a constraint solving problem using the Petri net model.

- **Automated Atomicity-violation Fixing** (2011) [34] is about AFix, whose repair model consists of putting instructions into critical regions.

4.3 Automated Repair of Build Scripts

- **Shipwright: A Human-in-the-Loop System for Dockerfile Repair** (2021,) Henkel et al.[343] designs 13 rules for making automated repairs to Dockerfiles which cannot successfully build, in a data-driven manner.

- **Styler: Learning Formatting Conventions to Repair Checkstyle Errors** (2019) Madeiral et al. [249] propose to automatically repair Checkstyle formatting errors that break the build.

- **History-driven build failure fixing: how far are we?** (2019,Ω) You et al. [251] show that a simple approach works better than HireBuild [171] on a new dataset of 102 reproducible Gradle build failures.

- **HireBuild: an automatic approach to history-driven repair of build scripts** (2018) [171] mines and apply build-fix patterns in Gradle, and apply them based on log similarity.

4.4 Repair for the Web

- **Usability and Aesthetics: Better Together for Automated Repair of Web Pages** (2021, ∀) Le-Cong et al. [336] design a meta-heuristic algorithm that evolves buggy web pages to optimize both usability and aesthetics.

- **Automated Repair of Cross-Site Scripting Vulnerabilities through Unit Testing** (2020,) Mohammadi et al. [259] automatically add calls to sanitizers to fix statically found XSS vulerabilities.

- **Fully Automated HTML and Javascript Rewriting for Constructing a Self-healing Web Proxy** (2018) [166] 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) [187] 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) [188] fixes web rendering by changing the value of CSS properties

- **Vejovis: Suggesting fixes for JavaScript faults** (2014) [72] suggests fixes for DOM errors based on fix patterns

- **Fix Me Up: Repairing Access-Control Bugs in Web Applications.** (2013) [61] 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** (2012) [47] fixes incorrect opening/closing HTML tags in PHP application by encoding the problem as string constraints.

4.5 Repair of Software Models

- **Transforming abstract to concrete repairs with a generative approach of repair values** (2021, 🕸) Kretschmer et al. [348] repair inconsistencies in UML models.
- **ARepair: a repair framework for Alloy** (2019,¤) Wang et al. [270] describe a generate-and-validate repair technique for Alloy models, with a test-based specification based on AUnit.
- **Range Fixes: Interactive Error Resolution for Software Configuration** [100] (2015) focuses on automatically repairing configuration errors in software product lines
- **Towards Automated Inconsistency Handling in Design Models** (2010) Silva et al. [28] use Prolog to propose a repair plan that fixes inconsistencies in UML models
- **Supporting Automatic Model Inconsistency Fixing** [22] (2009) detects and fixes inconsistencies in MOF and UML models
- **Repairing Unsatisfiable Concepts in OWL Ontologies** [7] (2006) states an automatic repair problem in the context of OWL ontologies.
- **Consistency Management with Repair Actions** [2] (2003) detects inconsistencies in XML documents and proposes repair actions accordingly.

4.6 Repair of Security Vulnerabilities

- **Example-Based Vulnerability Detection and Repair in Java Code** (2022, ∀) Zhang et al. [400] devise an approach where security experts first define a dataset of pairs of insecure/secure Java code (28 pairs in the experiment), and then an algorithm extracts the matching and fixing transformation.
- **Automatically Mitigating Vulnerabilities in x86 Binary Programs via Partially Recompilable Decompilation** (2022, ∀) Under the assumption that no source code is avilable, Reiter et al. [393] prove the feasibility of decompiling small chunks of code (using Hex-Rays), running GenProg on them, recompiling and reinjecting the fixed code in the binary to be executed.
- **Neural Transfer Learning for Repairing Security Vulnerabilities in C Code** (2021, 🕸) Chen et al. [335] train a transformer on bug-fixing commits and fine-tune it on real CVE vulnearbilities, proving that transfer learning happens between bug fixing and vulnerability fixing (the previous iteration was [223]).
- **Using Safety Properties to Generate Vulnerability Patches** (2019,¤) Huang et al. [236] generate check-and-error patches for buffer overflows, bad casts and integer overflows triggered by exploits and fuzzing inputs.
- **VuRLE - Automatic Vulnerability Detection and Repair by Learning from Examples** (2017) Ma et al. [132] learns systematic edits from examples and apply them to fix vulnerabilities in Android applications.
- **Cdrep: Automatic repair of cryptographic misuses in android applications** (2016, ⋕) Ma et al. [115] define 7 binary transformations for Dalvik bytecode to repair 7 cryptographic API misuses in Android.
- **AutoPaG: Towards Automated Software Patch Generation with Source Code Root Cause Identification and Repair** (2007) [9] generates a source code patch from an input that triggers an array overflow in C code, with failure-oblivious repair operators (adding a modulo in the read expression and truncating data to be written).
- **Countering Network Worms Through Automatic Patch Generation** (2005 [4] detect buffer overflow vulnerabilities at runtime in production, then produce a source code patch that skip the execution of the overflowing statement.

4.7 Repair of Smart Contracts

- **Elysium: Automagically Healing Vulnerable Smart Contracts Using Context-Aware Patching** (2021, ∀) Torres et al. [370] improve over Smartshield and SGuard by means of better code analysis and more automation.

- **SGuard: Towards Fixing Vulnerable Smart Contracts Automatically** (2021, ∀) Nguyen et al. [355] repair reentrancy and arithmetic bugs in smart contracts, at the source code level, with guarantees based on an operational semantics of Ethereum opcodes.

- **EVMPatch: Timely and automated patching of ethereum smart contracts** (2021, ∀) Rodler et al. [366] design an end-to-end technique to binary patch, back-test and deploy via delegation Ethereum smart contracts. The evaluation focuses on integer overflow attacks.

- **Smartshield: Automatic smart contract protection made easy** (2021, ∀) Zhang et al. [325] propose a smart contract binary transformation for 3 pattern based problems (state change after external calls, missing checks for out-of-bound arithmetic operations, and missing checks for failing external calls).

- **Smart Contract Repair** (2019, ★) Yu et al. [278] repair smart contracts in Ethereum to minimize gas consumption.

4.8 Misc Repair Types

- **PGPATCH: Policy-Guided Logic Bug Patching for Robotic Vehicles** (2022, ∀) Kim et al. [388] devise an end-to-end repair approach for robotic vehicle control programs (ArduPilot and PX4) based on 5 repair templates.

- **Automatic Repair for Network Programs** (2022, ∀) Shi et al [394] perform repair of programs in Floodlight, an open-source SDN controller based on Java annotations, using a domain-specific symbolic fault localization algorithm and enumerative synthesis.

- **CirFix: Automatically Repairing Defects in Hardware Design Code** (2022, ∀) Ahmad et al. [381] present a framework for automatically repairing defects in Verilog, based on a novel dataflow-based fault localization approach tailored for hardware description languages.

- **Automated Repair of Size-Based Inaccessibility Issues in Mobile Applications** (2021, ∀) Alotaibi et al. [328] develop an approach that automatically increases the size of Android UI elements, chosen based on a multi-objective minimization function.

- **Automatic repair of timestamp comparisons** (2021, ✪) Liva et al. [248] statically identify time comparison problems in programs and rewrite time comparison expressions in a safe normal form.

- **CRNRepair: Automated Program Repair of Chemical Reaction Networks** (2021, ✪) Mesecan et al. [353] bridge the GI framework PyGGI and the Matlab environment SimBiology to do original experiments on automated repair of chemical reaction networks.

- **Repairing serializability bugs in distributed database programs via automated schema refactoring** (2021, ✪) Rahmani et al. [364] target the problem of repairing transaction serializability bugs in databases.

- **TFix+: Self-configuring Hybrid Timeout Bug Fixing for Cloud Systems** (2021, ✪) He et al. [342] propose a technique to automatically repair timeout bugs in distributed systems such as Hadoop.

- **Automatic Software Merging using Automated Program Repair** (2019) [275] fixes merge conflicts with a search-based approach based on kGenProg.
- **Efficient Automated Repair of High Floating-Point Errors in Numerical Libraries** (2019,¤) Yi et al. [277] for numerical functions (eg from GNU Scientific Library), identify small parts of the input domain that have high floating point instability, and replace the original implementation by a better approximation.
- **Towards Specification-Directed Program Repair** (2018) [197] does program repair for the educational programming language Karel, by training a neural net to predict the edit (keep, delete, insert or replace token).
- **Automated model repair for Alloy** (2018) [209] does repair for the Alloy language with 11 mutation operators,
- **Caramel: Detecting and fixing performance problems that have non-intrusive fixes** (2015) Nistor et al. [95] presents a technique to suggest addition of 'break' statement guarded by a synthesized condition.
- **Automated Repair of High Inaccuracies in Numerical Programs** (2017) Yi et al. [154] use mathematically equivalent floating-point expressions that reduce inaccuracies found with random testing.
- **Data-guided Repair of Selection Statements** (2014) [64] repairs database selection statements in a specific data-oriented language (Abap fro SAP).
- **A Framework for the Automatic Correction of Constraint Programs** (2011) [37] repairs constraint programs the repair consisting of declaratively removing or adding new constraints.

4.9 SQL Repair

- **SQLRepair: Identifying and Repairing Mistakes in Student-Authored SQL Queries** (2021,) Presler et al.'s SQLRepair [362] combine heuristics and a SMT-based repair approach to fix SQL queries (tool at `https://github.com/kpresler/sqlrepair`).
- **Using Automated Fix Generation to Secure SQL Statements** (2007) Thomas et al. [10] describe an automatic transformation in Java for going from plain java SQL to prepared statements.

5 Optimization & Integration

5.1 Driving the Search

- **Multiplicative Weights Algorithms for Parallel Automated Software Repair** (2021,) Renzullo et al. [365] propose to use online learning based on multiplicative weights update to efficiently find those combinations of mutations which repair a bug.
- **Concolic Program Repair** (2021,) Shariffdeen et al.'s technique [367] consists of alternating patch enumeration, input synthesis and concolic execution on the synthesized input to generate a small amount of patches.
- **How Does Regression Test Selection Affect Program Repair? An Extensive Study on 2 Million Patches** (2021,) Lou et al. [351] claim that regression test selection is useful for program repair, based on experiments on Defects4J.
- **Self-Boosted Automated Program Repair** (2021,) Benton et al. [330] constantly re-order the patch candidate list to be verified wrt the test suite in order to speed up the discovery of plausible patches.

- **Leveraging Program Invariants to Promote Population Diversity in Search-Based Automatic Program Repair** (2019) [226] explores the usege of learned invariants to improve the fitness function of generate-and-validate program repair, experimenting with genprog4java.

- **A new word embedding approach to evaluate potential fixes for automated program repair** (2018) [157] computes source code line embeddings from word2vec embeddings in order to calculate distances between patches.

5.2 Addressing the patch overfitting problem

- **Identifying Incorrect Patches in Program Repair Based on Meaning of Source Code** (2022) Phung et al. [392] order APR patches by their distance to the method intention, where the method intention is inferred from the patched method name and the distance is computed in an embedding space based on Code2Vec.

- **Exploring Plausible Patches Using Source Code Embeddings in JavaScript** (2021) Csuvik et al. [338] present experiments suggesting that the Doc2Vec embedding of code is not useful for discarding overfitting patches.

- **Exploring True Test Overfitting in Dynamic Automated Program Repair using Formal Methods** (2021) Nilizadeh et al. [357] assess overfitting in APR patches using ground truth reference programs equipped with formal specifications in OpenJML, with an experiment on 30 small programs. Follow-up paper is [358].

- **Neural Program Repair with Execution-based Backpropagation** (2021) Ye et al. [377] design and optimize a loss function that embeds the results of test execution in order to avoid overfitting.

- **Adversarial Patch Generation for Automatic Program Repair** (2020) Alhefdhi et al. [283] present preliminary results on using a patch discriminator to encourage a data-driven system to generate patches that look like human patches.

- **Automated Patch Correctness Assessment: How Far are We?** (2020) Wang et al. [318] compare different overfitting detection techniques from the literature and find that dynamic techniques do not perform better than static techniques.

- **Evaluating representation learning of code changes for predicting patch correctness in program repair** (2020) Tien et al. [316] show that a purely syntactic approach based on BERT-based embeddings associated with logistic regression does not improve overfitting detection.

- **Exploring the Differences between Plausible and Correct Patches at Fine-Grained Level** (2020) Yang et al. [321] present a preliminary experiment on using Daikon invariants to detect overfitting patches.

- **Utilizing Source Code Embeddings to Identify Correct Patches** (2020) Csuvik et al. [292] propose to order likely patches by their distance to the buggy program in an embedding space, and compare three such spaces.

- **Automated Classification of Overfitting Patches with Statically Extracted Code Features** (2019, ★) Ye et al. [378] define features on code and train a machine learning model to detect overfitting patches.

- **Validation of Automatically Generated Patches: An Appetizer** (2019, ★) Ghanbari [231] proposes to use Daikon invariants to generate property-based tests that can rank generated patches by likelihood.

- **Automated Patch Assessment for Program Repair at Scale** (2019, ★) Ye et al. [376] studies the usage of test generation based on a ground truth patch to better evaluate program repair research.

- **Alleviating Patch Overfitting with Automatic Test Generation: A Study of Feasibility and Effectiveness for the Nopol Repair System** (2018) [216] shows that using tests that are generated against the buggy version of the program under repair poses a serious oracle problem.
- **Identifying Patch Correctness in Test-Based Program Repair** (2018) Xiong et al. [214] analyze test execution traces to filter out incorrect overfitting patches.
- **Overfitting in semantics-based automated program repair** (2018) [179] compares Angelix and variants of it on the IntroClass and CodeFlaws benchmarks showing that 50-75% of patches are overfitting.
- **Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair** (2015) [98] is the first paper to name the overfitting problem.

5.3 Improvement of the Fault Localization Step

- **Revisiting Test Cases to Boost Generate-and-Validate Program Repair** Zhang et al. [379] study how stacktraces can be be used to improve fault localization in APR.
- **On the effectiveness of unified debugging: An extensive study on 16 program repair systems** (2020,) Benton et al. [284] study the performance of a new fault localization technique called UniDebug++, on 16 repair tools. On Defects4J, UniDebug++ can localize over 20% more bugs at the Top-1 position.
- **Automatically Repairing Programs Using Both Tests and Bug Reports** (2020,), Motwani and Brun [312] improve on the fault localization component of SimFix with a new technique that combines spectrum-based and bug-report based fault localization.
- **Can Automated Program Repair Refine Fault Localization** (2019, ★) Lou et al. [250] proposes a variant of mutation-based fault localization based on the PraPR program repair tool.
- **Restore: Retrospective fault localization enhancing automated program repair** (2020,), Xu et al. [320] design a two-phase fault-localization process for repair and apply it to Jaid and SimFix.
- **You Cannot Fix What You Cannot Find! An Investigation of Fault Localization Bias in Benchmarking Automated Program Repair Systems** (2019) [247] shows that one third of bugs in the Defects4J benchmark cannot be localized, hence cannot be repair with approach based on spectrum-based fault localization.
- **An Empirical Study on the Effect of Dynamic Slicing on Automated Program Repair Efficiency** (2018) [168] replaces Ochiai in Nopol [149] by a dynamic slicing approach based on Javaslicer.
- **An Empirical Study on the Usage of Fault Localization in Automated Program Repair**[150] (2017) compares two variations of spectrum based fault localization in Nopol [149].

5.4 Interactive Program Repair

"Interactive Program Repair" means asking questions to the developer about the expected output of some expressions, in order to drive the search towards correct oatches.

- **Automatic Program Repair as Semantic Suggestions - An Empirical Study** (2021,) Campos et al. [333] implement and evaluate mutation-based repair for Javascript inside Visual Studio.

- **Interactive Patch Filtering as Debugging Aid** (2020, ⋕) Liang et al. develop an IDE plugin to present APR patches to developers in a debugging session and shows how it helps fixing the bug at hand, in an experiment over 30 students and 85 Defects4J bugs.

- **Human-In-The-Loop Automatic Program Repair** (2020, ⋕) Böhme et al. [286] propose to ask a fixed number of yes/no questions to the user/developer about the expected behavior of the program under repair in order to reduce the risk of incorrect patches.

- **Interactive Testing and Repairing of Regular Expressions** (2018) [159] proposes an interactive technique to repair regular expressions, the developer being asked for validation.

- **At the End of Synthesis: Narrowing Program Candidates** (2017) Shriver et al. [139] identify inputs on which the behavior of two candidate patches differ, and show them to the developers to ask about the preferred behavior.

5.5 Repair Speed

- **Program Repair with Repeated Learning** (2022, ∀) Chen et al. [382] propose a repair loop for generate-and-validate repair where a prioritization model is learned on the fly. The prioritization model is a learning-to-rank version of XGBoost, using 17 code features extracted from the patch, and which is updated depending on the compilation and test outcome (tool).

- **Speeding up constraint-based program repair using a search-based technique** (2022, ∀) Yi et al. [397] replace Angelix′ symbolic execution by Monte Carlo sampling over paths in order to find angelic paths.

- **Speedup automatic program repair using dynamic software updating: an empirical study** (2019) Guo et al. [232] apply generated patches using hotswapping / class reload in the JVM and report the presence of a speed-up.

- **Fast and Precise On-the-fly Patch Validation for All** (2020, ϒ) Chen and Zhang [289] propose to only load the tentatively patched binary Java classes through hot-swaping technology, in order to speed up validation with the test suite.

- **Test-equivalence Analysis for Automatic Patch Generation** [192] (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** (2017) [138] 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** [62] (2013) discards some repair candidates using program equivalent checks typical from compilers.

- **Efficient Automated Program Repair Through Fault-Recorded Testing Prioritization** [59] (2013) blends test suite prioritization and classical Genprog.

- **More Efficient Automatic Repair of Large-scale Programs Using Weak Recompilation** [45] (2012) creates an incremental compilation system that is dedicated to program repair.

5.6 Integration / UI / Tooling

- **On the introduction of automatic program repair in Bloomberg** (2021, 🕸) Kirbas et al. [346] describe the integration of APR at Bloomberg, with a system based on mining repair templates with anti-unification.
- **E-APR: Mapping the Effectiveness of Automated Program Repair** (2020, ∺) Aleti and Martinez [282] present a meta-tool to predict the right repair tool to use based on features of the buggy program.
- **Visualizing Code Genealogy: How Code is Evolutionarily Fixed in Program Repair** (2019, ★) Tomida et al. [267] proposes a user-interface to visualize the search happening in a generate-and-validate repair loop implemented in kGenProg.
- **Towards s/engineer/bot: principles for program repair bots** (2019,¤) van Tonder and Le Goues [268] discuss six principles for engineering repair bots related to syntax, semantics and integration.
- **SapFix: Automated End-to-End Repair at Scale** (2019) [255] 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) [207] is the first ever blueprint architecture on using program repair in continuous integration.
- **Synergistic Debug-Repair of Heap Manipulations** (2017, ★) Verma and Roy [143] add advanced concepts in a proof-of-concept debugger on top of GDB, which supports specifying desired states and patch generation via SMT-based repair constraints.
- **Should fixing these failures be delegated to automated program repair?** (2015) Le et al. [89] perform automatic classification of successful and unsuccessful cases in Genprog based on features from the Genprog search.

6 Position Papers

- **Explainable Software Bot Contributions: Case Study of Automated Bug Fixes** (2019) Monperrus [261] claims that patches generated with automatic program repair should come with a textual explanation.
- **Beyond testing configurable systems: applying variational execution to automatic program repair and higher order mutation testing** (2018) [213] 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) Weimer et al. [121]'s position paper is about detecting behavioral differences between patches using targeted differential testing.
- **When App Stores Listen to the Crowd to Fight Bugs in the Wild** (2015) [85] 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) [71] 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) [57] 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) [53] shows the vision of C. Le Goues at the end of her seminal PhD thesis on GenProg.

7 Formal Approaches to Program Repair

- **Automated Repair of Heap-Manipulating Programs using Deductive Synthesis** (2020, ♈), Nguyen et al. [356] fix static warnings found with HIP/SLEEK (as [112]) using constraint solving on top of the Songbird prover and deductive synthesis.

- **Deductive Program Repair** (2015) Kneuss et al. [87] do program repair for a "purely functional subset of Scala", evaluated on seeded bugs on small programs.

- **Cost-Aware Automatic Program Repair** (2014) [74] repairs boolean programs with assertions, by using the method of inductive assertions.

- **Program Repair As Sound Optimization of Broken Programs** (2009) [20] theoretically defines repair for an ad hoc formal language.

- **Program Repair Suggestions From Graphical State-Transition Specifications** (2008) [14] does theoretical repair using edit sequences on state machines.

- **Repair of Boolean Programs with An Application to C** (2006) [6] repairs a specific class of programs called boolean programs: those that only contain boolean variables.

- **Program Repair As a Game** (2005) [3] repair programs that are expressed in linear temporal logics

8 Miscellaneous

8.1 Datasets & Benchmarks

- **Is the Ground Truth Really Accurate? Dataset Purification for Automated Program Repair** (2021, ⛬) Yang et al. [373] use coverage and delta-debugging to perform change minimization of benchmark bugs (minimized D4J patches are at DehengYang/dataset_purification).

- **Towards a Benchmark Set for Program Repair Based on Partial Fixes** (2021, ⛬) Beyer et al. [332] curated 2204 benchmark tasks where the input is a partial fix (data at `https://gitlab.com/sosy-lab/software/partial-fix-benchmarks/`).

- **A critical review on the evaluation of automated program repair systems** (2020, ♈) Liu et al. [350] discuss and consolidate 8 evaluation metrics for program repair research, which cover different aspects of the problem space.

- **Critical Review of BugSwarm for Fault Localization and Program Repair** (2019,¤) Durieux et al. [228] state desirable properties applying to benchmarks for program repair and assess BugSwarm according to them, showing that a minority of bugs are usable in this context.

- **BugSwarm: Mining and Continuously Growing a Dataset of Reproducible Failures and Fixes** (2019) [227] uses Travis CI as [253] 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. [253] propose a new benchmark whose novelty is to be based on continuous integration analysis (and not on past commits).
- **DroidBugs: An Android Benchmark for Automated Program Repair** (2018) Azevedo et al. [160] gathers 13 bugs in Android apps. (code)
- **Bugs.jar: a large-scale, diverse dataset of real-world Java bugs** (2018) [195] describes a dataset of 1,158 bugs and patches, over 8 open-source projects.
- **Codeflaws: a programming competition benchmark for evaluating automated program repair tools** (2017) Tan et al. [140] present a benchmark of 3902 defects in C, crawled from the Codeforces programming competition website.
- **QuixBugs: a multi-lingual program repair benchmark set based on the quixey challenge** (2017) [130] is a benchmark of in 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 [90] 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 [104]).
- **Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs** (2014) Just et al. [65] presents the Defects4J benchmark, extensively used in program repair research since the initial experiment by Durieux et al. [81, 133].

8.2 Automatic Hardening

- **Automatically Fixing C Buffer Overflows Using Program Transformations** (2014) [75] uses three program transformations dedicated to integer operations, and shows that the approach scales to real programs.
- **Program Transformations to Fix C Integers** (2013) [49] 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) [50] automatically adds a condition checks after all method calls with a source-to-source transformation in C code.

8.3 Surveys

- **Neural Program Repair: Systems, Challenges and Solutions** (2022, ∀) Zhong et al. [401] surveys the works on using neural networks to generate patches (51 references).
- **A Survey on Automatic Bug Fixing** (2020, Ψ), Cao et al. [287] summarize the recent advances made since the previous surveys (113 references).
- **Automated Program Repair** (2019) Le Goues et al. [242] give a high-level view of the field in Communications of the ACM (40 references).
- **A Survey of Test Based Automatic Program Repair** (2018) Liu et al. [185] present 81 references, with the last ones from 2017.
- **Automatic software repair: a Survey** (2017) Gazzola et al.'s survey [127] at IEEE TSE with 176 references.
- **Automatic software repair: a Bibliography** Monperrus [134] (first online, 2015, journal version 2017) is the first ever survey of the field, with 197 references.

8.4 Doctoral Theses

- Zakharchenko, "A practical approach to automated software correctness enhancement", 2022 [398]
- Chen, "Effective automatic program repair based on state abstraction", 2021 [334]
- Gao, "Overfitting in Program Repair and Synthesis", 2021 [340]
- Ginelli, "Understanding and Improving Automatic Program Repair: A Study of Code-removal Patches and a New Exception-driven Fault Localization Approach", 2020 [297]
- Koyuncu, "Boosting Automated Program Repair for Adoption By Practitioners", 2020 [304]
- Coker, "Automatic Repair of Framework Applications", 2020 [291]
- Harer, "Improved neural machine translation systems for low resource correction tasks", 2019 [234]
- Liu, "Deep Pattern Mining for Program Repair", 2018 [182]
- Durieux, "From Runtime Failures to Patches: Study of Patch Generation in Production", 2018 [165]
- Le, "Overfitting in Automated Program Repair: Challenges and Solutions", 2018 [178]
- Long, "Automatic patch generation via learning from successful human patches", 2018 [186]
- Hua, "Unifying Program Repair and Program Synthesis", 2018 [174]
- Mechtaev, "Semantic Program Repair", 2018 [190]
- Tan, "Design of repair operators for automated program repair", 2018 [201]
- Timperley, "Advanced Techniques for Search-Based Program Repair", 2017 [142]
- Gopinath, "Systematic techniques for more effective fault localization and program repair", 2016 [105]
- Cornu, "Automatic Analysis and Repair of Exception Bugs for Java Programs", 2015 [79]
- Martinez, "Extraction and Analysis of Knowledge for Automatic Software Repair", 2014 [69]
- Le Goues, "Automatic Program Repair Using Genetic Programming", 2013 [52]
- Nguyen, "Automating Program Verification and Repair Using Invariant Analysis and Test Input Generation", 2010 [26]
- Arcuri, "Automatic Software Generation and Improvement Through Search Based Techniques", 2009 [15]

References

[1] Rafael Corchuelo et al. "Repairing syntax errors in LR parsers". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 24.6 (2002), pp. 698–710.

[2] Christian Nentwich, Wolfgang Emmerich, and Anthony Finkelstein. "Consistency Management with Repair Actions". In: *Proceedings of the 25th International Conference on Software Engineering*. 2003, pp. 455–464.

[3] Barbara Jobstmann, Andreas Griesmayer, and Roderick Bloem. "Program Repair As a Game". In: *Computer Aided Verification*. 2005, pp. 226–238.

[4] S. Sidiroglou and A.D. Keromytis. "Countering Network Worms Through Automatic Patch Generation". In: *Security & Privacy* 3.6 (2005), pp. 41–49.

[5] Louise A. Dennis, Raul Monroy, and Pablo Nogueira. "Proof-directed Debugging and Repair". In: *Seventh Symposium on Trends in Functional Programming*. 2006, pp. 131–140.

[6] Andreas Griesmayer, Roderick Bloem, and Byron Cook. "Repair of Boolean Programs with An Application to C". In: *Computer Aided Verification*. 2006, pp. 358–371.

[7] Aditya Kalyanpur et al. "Repairing Unsatisfiable Concepts in OWL Ontologies". In: *The Semantic Web: Research and Applications*. Vol. 4011. 2006, pp. 170–184.

[8] Westley Weimer. "Patches As Better Bug Reports". In: *Proceedings of the International Conference on Generative Programming and Component Engineering*. 2006.

[9] Z. Lin et al. "AutoPaG: Towards Automated Software Patch Generation with Source Code Root Cause Identification and Repair". In: *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*. 2007, pp. 329–340.

[10] S. Thomas and L. Williams. "Using Automated Fix Generation to Secure SQL Statements". In: *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*. 2007, p. 9.

[11] Andrea Arcuri. "On the Automation of Fixing Software Bugs". In: *Companion of the 30th International Conference on Software Engineering*. 2008, pp. 1003–1006.

[12] Andrea Arcuri and Xin Yao. "A Novel Co-evolutionary Approach to Automatic Software Bug Fixing". In: *Proceedings of the IEEE Congress on Evolutionary Computation*. 2008, pp. 162–168.

[13] Atif M. Memon. "Automatically Repairing Event Sequence-based GUI Test Suites for Regression Testing". In: *ACM Transactions on Software Engineering and Methodology* 18.2 (2008), p. 4.

[14] Farn Wang and Chih-Hong Cheng. "Program Repair Suggestions From Graphical State-Transition Specifications". In: *Proceedings of FORTE 2008*. 2008.

[15] Andrea Arcuri. "Automatic Software Generation and Improvement Through Search Based Techniques". PhD thesis. University of Birmingham, 2009.

[16] Valentin Dallmeier, Andreas Zeller, and Bertrand Meyer. "Generating Fixes From Object Behavior Anomalies". In: *Proceedings of the International Conference on Automated Software Engineering*. 2009.

[17] Brett Daniel et al. "ReAssert: Suggesting Repairs for Broken Unit Tests". In: *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*. 2009, pp. 433–444.

[18] S. Forrest et al. "A Genetic Programming Approach to Automated Software Repair". In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. 2009, pp. 947–954.

[19] Dennis Jeffrey et al. "BugFix: a Learning-based Tool to Assist Developers in Fixing Bugs". In: *ICPC*. 2009, pp. 70–79.

[20] Yuhua Qi, Xiaoguang Mao, and Yan Lei. "Program Repair As Sound Optimization of Broken Programs". In: *International Symposium on Theoretical Aspects of Software Engineering*. 2009.

[21] W. Weimer et al. "Automatically Finding Patches Using Genetic Programming". In: *Proceedings of the International Conference on Software Engineering*. 2009.

[22] Yingfei Xiong et al. "Supporting Automatic Model Inconsistency Fixing". In: *7th joint meeting of the European Software engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*. ACM. 2009, pp. 315–324.

[23] V. Debroy and W.E. Wong. "Using Mutation to Automatically Suggest Fixes for Faulty Programs". In: *Proceedings of the International Conference on Software Testing, Verification and Validation*. 2010, pp. 65–74.

[24] Ethan Fast et al. "Designing Better Fitness Functions for Automated Program Repair". In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. 2010, pp. 965–972.

[25] Christian Kern and Javier Esparza. "Automatic Error Correction of Java Programs". In: *Formal Methods for Industrial Critical Systems*. 2010, pp. 67–81.

[26] Thanh V. Nguyen. "Automating Program Verification and Repair Using Invariant Analysis and Test Input Generation". PhD thesis. The Unversity of New Mexico, 2010.

[27] E. Schulte, S. Forrest, and W. Weimer. "Automated Program Repair Through the Evolution of Assembly Code". In: *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. 2010, pp. 313–316.

[28] Marcos Aurélio Almeida da Silva et al. "Towards Automated Inconsistency Handling in Design Models". In: *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering*. 2010, pp. 348–362.

[29] Yi Wei et al. "Automated Fixing of Programs with Contracts". In: *Proceedings of the International Symposium on Software Testing and Analysis*. 2010.

[30] Westley Weimer et al. "Automatic Program Repair with Evolutionary Computation". In: *Communications of the ACM* 53.5 (2010), p. 109.

[31] T. Ackling, B. Alexander, and I. Grunert. "Evolving Patches for Software Repair". In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. 2011, pp. 1427–1434.

[32] Andrea Arcuri. "Evolutionary Repair of Faulty Software". In: *Applied Soft Computing* 11.4 (2011), pp. 3494–3514.

[33] Divya Gopinath, Muhammad Zubair Malik, and Sarfraz Khurshid. "Specification-based Program Repair Using SAT". In: *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2011.

[34] G. Jin et al. "Automated Atomicity-violation Fixing". In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2011, pp. 389–400.

[35] Sara Kalvala and Richard Warburton. "A Formal Approach to Fixing Bugs". In: *Formal Methods, Foundations and Applications*. 2011, pp. 172–187.

[36] Robert Könighofer and Roderick Bloem. "Automated Error Localization and Correction for Imperative Programs". In: *Formal Methods in Computer-Aided Design (FMCAD), 2011*. 2011, pp. 91–100.

[37] N. Lazaar, A. Gotlieb, and Y. Lebbah. "A Framework for the Automatic Correction of Constraint Programs". In: *Proceedings of the International Conference on Software Testing, Verification and Validation*. 2011, pp. 319–326.

[38] M.Z. Malik, J.H. Siddiqi, and S. Khurshid. "Constraint-Based Program Debugging Using Data Structure Repair". In: *International Conference on Software Testing, Verification and Validation (ICST)*. 2011, pp. 190–199.

[39] Yu Pei et al. *Code-Based Automated Program Fixing*. Tech. rep. arXiv:1102.1059v2. 2011.

[40] Zachary P. Fry, Bryan Landau, and Westley Weimer. "A Human Study of Patch Maintainability". In: *Proceedings of the International Symposium on Software Testing and Analysis*. 2012, pp. 177–187.

[41] C. Le Goues et al. "A Systematic Study of Automated Program Repair: Fixing 55 Out of 105 Bugs for $8 Each". In: *Proceedings of the International Conference on Software Engineering*. 2012, pp. 3–13.

[42] Claire Le Goues et al. "GenProg: a Generic Method for Automatic Software Repair". In: *IEEE Transactions on Software Engineering* 38 (2012), pp. 54–72.

[43] Peng Liu and Charles Zhang. "Axis: Automatically Fixing Atomicity Violations Through Solving Control Constraints". In: *Proceedings of the 2012 International Conference on Software Engineering*. 2012, pp. 299–309.

[44] Francesco Logozzo and Tom Ball. "Modular and Verified Automatic Program Repair". In: *Proceedings of the 27th ACM International Conference on Object Oriented Programming Systems Languages and Applications*. 2012.

[45] YuHua Qi et al. "More Efficient Automatic Repair of Large-scale Programs Using Weak Recompilation". In: *Science China Information Sciences* 55.12 (2012), pp. 2785–2799.

[46] Urmas Repinski et al. "Combining dynamic slicing and mutation operators for ESL correction". In: *17th IEEE European Test Symposium*. IEEE. 2012, pp. 1–6.

[47] Hesam Samimi et al. "Automated Repair of HTML Generation Errors in PHP Applications Using String Constraint Solving". In: *Proceedings of ICSE*. 2012, pp. 277–287.

[48] Vipin Balachandran. "Fix-it: An extensible code auto-fix component in review bot". In: *IEEE 13th International Working Conference on Source Code Analysis and Manipulation, SCAM 2013*. 2013, pp. 167–172.

[49] Zack Coker and Munawar Hafiz. "Program Transformations to Fix C Integers". In: *Proceedings of the International Conference on Software Engineering*. 2013, pp. 792–801.

[50] Youry Khmelevsky, Martin C Rinard, and Stelios Sidiroglou-Douskos. "A Source-to-source Transformation Tool for Error Fixing." In: *Proceedings of CASCON*. 2013, pp. 147–160.

[51] Dongsun Kim et al. "Automatic Patch Generation Learned From Human-Written Patches". In: *Proceedings of ICSE*. 2013.

[52] Claire Le Goues. "Automatic Program Repair Using Genetic Programming". PhD thesis. University of Virgina, 2013.

[53] Claire Le Goues, Stephanie Forrest, and Westley Weimer. "Current Challenges in Automatic Software Repair". In: *Software Quality Journal* 21.3 (2013), pp. 421–443.

[54] Maurizio Leotta et al. "Repairing Selenium Test Cases: an Industrial Case Study about Web Page Element Localization". In: *International Conference on Software Testing, Verification and Validation*. IEEE. 2013, pp. 487–488.

[55] Chen Liu et al. "R2Fix: Automatically Generating Bug Fixes From Bug Reports". In: *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST)*. 2013, pp. 282–291.

[56] Francesco Logozzo and Matthieu Martel. "Automatic Repair of Overflowing Expressions with Abstract Interpretation". In: *Semantics, Abstract Interpretation, and Reasoning About Programs: Essays Dedicated to David A. Schmidt on the Occasion of His Sixtieth Birthday*. Vol. 129. 2013, pp. 341–357.

[57] Martin Monperrus and Benoit Baudry. *Two Flavors in Automated Software Repair: Rigid Repair and Plastic Repair*. Research Report Dagstuhl Seminar 13061 "Fault Prediction, Localization, and Repair". Schloss Dagstuhl - Leibniz Center for Informatics, 2013, p. 5.

[58] Hoang Duong Thien Nguyen et al. "SemFix: Program Repair via Semantic Analysis". In: *Proceedings of the International Conference on Software Engineering*. 2013.

[59] Y. Qi, X. Mao, and Y. Lei. "Efficient Automated Program Repair Through Fault-Recorded Testing Prioritization". In: *Proceedings of ICSM*. 2013.

[60] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. "Automated Feedback Generation for Introductory Programming Assignments". In: *ACM SIGPLAN Notices*. Vol. 48. 6. 2013, pp. 15–26.

[61] Sooel Son, Kathryn S McKinley, and Vitaly Shmatikov. "Fix Me Up: Repairing Access-Control Bugs in Web Applications." In: *Proceedings of the Network and Distributed System Security Symposium*. 2013.

[62] Westley Weimer, Zachary P. Fry, and Stephanie Forrest. "Leveraging program equivalence for adaptive program repair: Models and first results". In: *International Conference on Automated Software Engineering*. 2013, pp. 356–366.

[63] Favio Demarco et al. "Automatic Repair of Buggy If Conditions and Missing Preconditions with SMT". In: *Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis (CSTVA 2014)*. 2014.

[64] Divya Gopinath et al. "Data-guided Repair of Selection Statements". In: *Proceedings of the 36th International Conference on Software Engineering*. 2014, pp. 243–253.

[65] René Just, Darioush Jalali, and Michael D. Ernst. "Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs". In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA'14)*. 2014.

[66] Shalini Kaleeswaran et al. "Minthint: Automated Synthesis of Repair Hints". In: *Proceedings of the International Conference on Software Engineering*. 2014, pp. 266–276.

[67] Yiyan Lin and Sandeep Kulkarni. "Automatic Repair for Multi-threaded Programs with Deadlock/Livelock Using Maximum Satisfiability". In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM. 2014, pp. 237–247.

[68] Fan Long et al. "Sound Input Filter Generation for Integer Overflow Errors". In: *ACM SIGPLAN Notices* 49.1 (2014), pp. 439–452.

[69] Matias Martinez. "Extraction and Analysis of Knowledge for Automatic Software Repair". PhD thesis. Université de Lille, 2014.

[70] Matias Martinez, Westley Weimer, and Martin Monperrus. "Do the Fix Ingredients Already Exist? An Empirical Inquiry into the Redundancy Assumptions of Program Repair Approaches". In: *ICSE - 36th IEEE International Conference on Software Engineering*. 2014.

[71] Martin Monperrus. "A Critical Review of "Automatic Patch Generation Learned from Human-Written Patches": Essay on the Problem Statement and the Evaluation of Automatic Software Repair". In: *International Conference on Software Engineering*. 2014, pp. 234–242.

[72] Frolin S Ocariza Jr, Karthik Pattabiraman, and Ali Mesbah. "Vejovis: suggesting fixes for JavaScript faults". In: *Proceedings of the 36th International Conference on Software Engineering*. 2014.

[73] Yuhua Qi et al. "The Strength of Random Search on Automated Program Repair". In: *Proceedings of the 36th International Conference on Software Engineering*. 2014, pp. 254–265.

[74] Roopsha Samanta, Oswaldo Olivo, and E Allen Emerson. "Cost-aware Automatic Program Repair". In: *International Static Analysis Symposium*. Springer. 2014, pp. 268–284.

[75] Alex Shaw, Dusten Doggett, and Munawar Hafiz. "Automatically Fixing C Buffer Overflows Using Program Transformations". In: *International Conference on Dependable Systems and Networks*. 2014, pp. 124–135.

[76] Yida Tao et al. "Automatically Generated Patches As Debugging Aids: a Human Study". In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2014, pp. 64–74.

[77] Tielei Wang, Chengyu Song, and Wenke Lee. “Diagnosis and Emergency Patch Generation for Integer Overflow Exploits”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2014, pp. 255–275.

[78] Andreas Zeller et al. “Automated Fixing of Programs with Contracts”. In: *IEEE Transactions on Software Engineering* 40.5 (2014), pp. 427–449.

[79] Benoit Cornu. “Automatic Analysis and Repair of Exception Bugs for Java Programs”. PhD thesis. Université de Lille, 2015.

[80] Aritra Dhar et al. “CLOTHO: Saving Programs from Malformed Strings and Incorrect String-handling”. In: *Foundations of Software Engineering*. ACM. 2015, pp. 555–566.

[81] Thomas Durieux et al. *Automatic Repair of Real Bugs: An Experience Report on the Defects4J Dataset*. Tech. rep. hal-01162221. HAL, 2015.

[82] Qing Gao et al. “Fixing Recurring Crash Bugs via Analyzing Q&A Sites”. In: *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2015.

[83] Qing Gao et al. “Safe Memory-leak Fixing for C Programs”. In: *Proceedings of the 37th International Conference on Software Engineering*. 2015, pp. 459–470.

[84] Z. Gao et al. “SITAR: GUI Test Script Repair”. In: *IEEE Transactions on Software Engineering* (2015).

[85] Maria Gomez et al. “When App Stores Listen to the Crowd to Fight Bugs in the Wild”. In: *37th International Conference on Software Engineering (ICSE), track on New Ideas and Emerging Results (NIER)*. IEEE, 2015, p. 4.

[86] Yalin Ke et al. “Repairing Programs with Semantic Code Search”. In: *Proceedings of the International Conference on Automated Software Engineering*. 2015.

[87] Etienne Kneuss, Manos Koukoutos, and Viktor Kuncak. “Deductive Program Repair”. In: *International Conference on Computer Aided Verification*. Springer. 2015, pp. 217–233.

[88] Sebastian Lamelas and Martin Monperrus. *Automatic Repair of Infinite Loops*. Technical Report hal-01144026. University of Lille, 2015.

[89] Xuan-Bach D. Le, Tien-Duy B. Le, and David Lo. “Should fixing these failures be delegated to automated program repair?” In: *Proceedings of the IEEE International Symposium on Software Reliability Engineering*. 2015, pp. 427–437.

[90] Claire Le Goues et al. “The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs”. In: *IEEE Transactions on Software Engineering (TSE), in press* (2015).

[91] Fan Long and Martin C. Rinard. “Staged Program Repair with Condition Synthesis”. In: *Proceedings of ESEC/FSE*. 2015.

[92] Matias Martinez and Martin Monperrus. “Mining Software Repair Models for Reasoning on the Search Space of Automated Program Fixing”. In: *Empirical Software Engineering* 20.1 (2015), pp. 176–205.

[93] Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. “DirectFix: Looking for Simple Program Repairs”. In: *Proceedings of the 37th International Conference on Software Engineering*. 2015.

[94] P. Muntean et al. “Automated Generation of Buffer Overflows Quick Fixes Using Symbolic Execution and SMT”. In: *International Conference on Computer Safety, Reliability & Security (SAFECOMP’15)*. 2015.

[95] Adrian Nistor et al. “Caramel: Detecting and fixing performance problems that have non-intrusive fixes”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE. 2015, pp. 902–912.

[96] Renaud Pawlak et al. “Spoon: A Library for Implementing Analyses and Transformations of Java Source Code”. In: *Software: Practice and Experience* 46 (2015), pp. 1155–1179.

[97] Zichao Qi et al. “An Analysis of Patch Plausibility and Correctness for Generate-And-Validate Patch Generation Systems”. In: *Proceedings of ISSTA*. 2015.

[98] Edward K. Smith et al. “Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair”. In: *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 2015.

[99] Shin Hwei Tan and Abhik Roychoudhury. “Relifix: Automated Repair of Software Regressions”. In: *Proceedings of ICSE*. 2015.

[100] Yingfei Xiong et al. “Range Fixes: Interactive Error Resolution for Software Configuration”. In: *IEEE Transactions on Software Engineering* 41.6 (2015), pp. 603–619.

[101] Sahil Bhatia and Rishabh Singh. “Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks”. In: *arXiv* abs/1603.06129 (2016).

[102] Loris D’Antoni, Roopsha Samanta, and Rishabh Singh. “Qlose: Program Repair with Quantitative Objectives”. In: *International Conference on Computer Aided Verification*. Springer. 2016, pp. 383–401.

[103] Thomas Durieux and Martin Monperrus. “DynaMoth: Dynamic Code Synthesis for Automatic Program Repair”. In: *11th International Workshop in Automation of Software Test (AST 2016)*. 2016.

[104] Thomas Durieux and Martin Monperrus. *IntroClassJava: A Benchmark of 297 Small and Buggy Java Programs*. Research Report hal-01272126. Universite Lille 1, 2016.

[105] Divya Gopinath. “Systematic techniques for more effective fault localization and program repair”. PhD thesis. University of Texas, 2016.

[106] Sumit Gulwani, Ivan Radiček, and Florian Zuleger. “Automated Clustering and Program Repair for Introductory Programming Assignments”. In: *arXiv preprint arXiv:1603.03165* (2016).

[107] Mouna Hammoudi, Gregg Rothermel, and Andrea Stocco. “Waterfall: An incremental approach for repairing record-replay tests of web applications”. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. 2016, pp. 751–762.

[108] Tao Ji et al. “Automated Program Repair by Using Similar Code Containing Fix Ingredients”. In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)* 1 (2016), pp. 197–202.

[109] Mingyue Jiang et al. “A Metamorphic Testing Approach for Supporting Program Repair without the Need for a Test Oracle”. In: *Journal of Systems and Software* (2016).

[110] X. B. D. Le, D. Lo, and C. L. Goues. “History Driven Program Repair”. In: *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2016, pp. 213–224.

[111] X. D. Le, D. Lo, and C. Le Goues. “Empirical Study on Synthesis Engines for Semantics-Based Program Repair”. In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2016.

[112] Xuan-Bach D Le et al. “Enhancing automated program repair with deductive verification”. In: *IEEE International Conference on Software Maintenance and Evolution*. IEEE. 2016.

[113] Haopeng Liu, Yuxi Chen, and Shan Lu. “Understanding and Generating High Quality Patches for Concurrency bugs”. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. 2016, pp. 715–726.

[114] Fan Long and Martin C. Rinard. “Prophet: Automatic Patch Generation via Learning From Successful Patches”. In: *Proceedings of the Symposium on Principles of Programming Languages*. 2016.

[115] Siqi Ma et al. “Cdrep: Automatic repair of cryptographic misuses in android applications”. In: *Proceedings of the ACM on Asia Conference on Computer and Communications Security*. 2016.

[116] Matias Martinez and Martin Monperrus. “ASTOR: A Program Repair Library for Java”. In: *Proceedings of ISSTA, Demonstration Track*. 2016, pp. 441–444.

[117] Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. “Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis”. In: *Proceedings of the 38th International Conference on Software Engineering*. 2016, pp. 691–701.

[118] Vinicius Oliveira et al. “Improved Crossover Operators for Genetic Programming for Program Repair”. In: *Proceedings of the 8th International Symposium on Search Based Software Engineering*. 2016.

[119] Yewen Pu et al. “sk_p: a neural program corrector for MOOCs”. In: *Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity*. 2016.

[120] Bat-Chen Rothenberg and Orna Grumberg. “Sound and complete mutation-based program repair”. In: *International Symposium on Formal Methods*. 2016, pp. 593–611.

[121] W. Weimer et al. “Trusted Software Repair for System Resiliency”. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. 2016, pp. 238–241.

[122] Liushan Chen, Yu Pei, and Carlo A. Furia. “Contract-based Program Repair Without the Contracts”. In: *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*. 2017.

[123] Xi Cheng et al. “IntPTI: Automatic integer error repair with proper-type inference”. In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press. 2017, pp. 996–1001.

[124] Jacob Devlin et al. “Semantic Code Repair using Neuro-Symbolic Transformation Networks”. In: *arXiv preprint arXiv:1710.11054* (2017).

[125] Thomas Durieux, Youssef Hamadi, and Martin Monperrus. “Production-Driven Patch Generation”. In: *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*. 2017, pp. 23–26.

[126] Thomas Durieux et al. “Dynamic Patch Generation for Null Pointer Exceptions Using Metaprogramming”. In: *IEEE International Conference on Software Analysis, Evolution and Reengineering*. 2017, pp. 349–358.

[127] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. “Automatic software repair: A survey”. In: *IEEE Transactions on Software Engineering* (2017).

[128] Rahul Gupta et al. “DeepFix: Fixing Common C Language Errors by Deep Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2017.

[129] Xianglong Kong et al. “The impacts of techniques, programs and tests on automated program repair: An empirical study”. In: *Journal of Systems and Software* (2017).

[130] Derrick Lin et al. “QuixBugs: a multi-lingual program repair benchmark set based on the quixey challenge”. In: *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. 2017.

[131] Fan Long, Peter Amidon, and Martin Rinard. “Automatic Inference of Code Transforms for Patch Generation”. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 2017.

[132] Siqi Ma et al. “Vurle: Automatic vulnerability detection and repair by learning from examples”. In: *European Symposium on Research in Computer Security*. Springer. 2017, pp. 229–246.

[133] Matias Martinez et al. “Automatic Repair of Real Bugs in Java: A Large-Scale Experiment on the Defects4J Dataset”. In: *Empirical Software Engineering* 22.4 (2017), pp. 1936–1964.

[134] Martin Monperrus. “Automatic Software Repair: a Bibliography”. In: *ACM Computing Surveys* 51 (2017), pp. 1–24.

[135] Manish Motwani et al. “Do automated program repair techniques repair hard and important bugs?” In: *Empirical Software Engineering* (2017).

[136] ThanhVu Nguyen et al. “Connecting Program Synthesis and Reachability: Automatic Program Repair Using Test-Input Generation”. In: *TACAS*. 2017, pp. 301–318.

[137] Ripon K. Saha, Yingjun Lyu, and Hiroaki Yoshida. “Elixir: Effective object-oriented program repair”. In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. 2017.

[138] Liam Schramm. “Improving performance of automatic program repair using learned heuristics”. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM. 2017, pp. 1071–1073.

[139] David Shriver, Sebastian Elbaum, and Kathryn T. Stolee. “At the End of Synthesis: Narrowing Program Candidates”. In: *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*. 2017.

[140] Shin Hwei Tan et al. “Codeflaws: a programming competition benchmark for evaluating automated program repair tools”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE. 2017, pp. 180–182.

[141] Y. Tian and B. Ray. “Automatically Diagnosing and Repairing Error Handling Bugs in C”. In: *FSE’17*. 2017.

[142] Christopher Steven Timperley. “Advanced Techniques for Search-Based Program Repair”. PhD thesis. University of York, 2017.

[143] Sahil Verma and Subhajit Roy. “Synergistic debug-repair of heap manipulations”. In: *Proceedings of the Joint Meeting on Foundations of Software Engineering*. 2017.

[144] Ke Wang, Rishabh Singh, and Zhendong Su. “Dynamic Neural Program Embedding for Program Repair”. In: *arXiv preprint arXiv:1711.07163* (2017).

[145] Ming Wen et al. “An empirical analysis of the influence of fault space on search-based automated program repair”. In: *arXiv preprint arXiv:1707.05172* (2017).

[146] Martin White et al. *Sorting and Transforming Program Repair Ingredients via Deep Learning Code Similarities*. Tech. rep. 1707.04742. Arxiv, 2017.

[147] Qi Xin and Steven P Reiss. “Leveraging syntax-related code for automated program repair”. In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2017.

[148] Yingfei Xiong et al. “Precise Condition Synthesis for Program Repair”. In: *Proceedings of the 39th International Conference on Software Engineering*. 2017.

[149] Jifeng Xuan et al. “Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs”. In: *IEEE Transactions on Software Engineering* 43.1 (2017), pp. 34–55.

[150] D. Yang, Y. Qi, and X. Mao. “An Empirical Study on the Usage of Fault Localization in Automated Program Repair”. In: *2017 IEEE International Conference on Software Maintenance and Evolution*. 2017, pp. 504–508.

[151] Jinqiu Yang et al. “Better test cases for better automated program repair”. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.* 2017.

[152] J. Yi et al. “A feasibility study of using automated program repair for introductory programming assignments”. In: *Proceedings of ESEC/FSE.* 2017.

[153] Jooyong Yi et al. “A correlation study between automated program repair and test-suite metrics”. In: *Empirical Software Engineering* (2017).

[154] Xin Yi et al. “Automated Repair of High Inaccuracies in Numerical Programs”. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE. 2017, pp. 514–518.

[155] Yuan Yuan and Wolfgang Banzhaf. *ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming.* Tech. rep. 1712.07804. arXiv, 2017.

[156] Umair Z Ahmed et al. “Compilation error repair: for the student programs, from the student programs”. In: *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training.* ACM. 2018, pp. 78–87.

[157] Leonardo Afonso Amorim et al. “A new word embedding approach to evaluate potential fixes for automated program repair”. In: *2018 International Joint Conference on Neural Networks (IJCNN).* 2018, pp. 1–8.

[158] Gabin An, Jinhan Kim, and Shin Yoo. “Comparing line and AST granularity level for program repair using PyGGI”. In: *Proceedings of the 4th International Workshop on Genetic Improvement.* 2018, pp. 19–26.

[159] Paolo Arcaini, Angelo Gargantini, and Elvinia Riccobene. “Interactive Testing and Repairing of Regular Expressions”. In: *International Conference on Testing Software and Systems.* 2018, pp. 1–16.

[160] Larissa Azevedo, Altino Dantas, and Celso G Camilo-Junior. “DroidBugs: An Android Benchmark for Automatic Program Repair”. In: *arXiv preprint arXiv:1809.07353* (2018).

[161] S. Bhatia, P. Kohli, and R. Singh. “Neuro-Symbolic Program Corrector for Introductory Programming Assignments”. In: *2018 IEEE/ACM 40th International Conference on Software Engineering.* 2018, pp. 60–70.

[162] Zimin Chen and Martin Monperrus. *The CodRep Machine Learning on Source Code Competition.* Tech. rep. 1807.03200. arXiv, 2018.

[163] Zimin Chen and Martin Monperrus. *The Remarkable Role of Similarity in Redundancy-based Program Repair.* Tech. rep. 1811.05703. arXiv, 2018.

[164] Lukas Diekmann and Laurence Tratt. “Reducing Cascading Parsing Errors Through Fast Error Recovery”. In: *arXiv preprint arXiv:1804.07133* (2018).

[165] Thomas Durieux. “From Runtime Failures to Patches: Study of Patch Generation in Production”. PhD thesis. Université de Lille, 2018.

[166] Thomas Durieux, Youssef Hamadi, and Martin Monperrus. “Fully Automated HTML and Javascript Rewriting for Constructing a Self-healing Web Proxy”. In: *Proceedings of the 29th IEEE International Symposium on Software Reliability Engineering.* 2018.

[167] Ali Ghanbari and Lingming Zhang. “Practical Program Repair via Bytecode Mutation”. In: *arXiv* abs/1807.03512 (2018).

[168] Anbang Guo et al. “An Empirical Study on the Effect of Dynamic Slicing on Automated Program Repair Efficiency”. In: *International Conference on Software Maintenance and Evolution.* 2018, pp. 554–558.

[169] Rahul Gupta, Aditya Kanade, and Shirish Shevade. “Deep reinforcement learning for programming language correction”. In: *arXiv preprint arXiv:1801.10467* (2018).

[170] Jacob Harer et al. “Learning to repair software vulnerabilities with generative adversarial networks”. In: *Advances in Neural Information Processing Systems.* 2018, pp. 7933–7943.

[171] Foyzul Hassan and Xiaoyin Wang. “HireBuild: an automatic approach to history-driven repair of build scripts”. In: *Proceedings of the 40th International Conference on Software Engineering*. 2018, pp. 1078–1089.

[172] Hideaki Hata, Emad Shihab, and Graham Neubig. “Learning to Generate Corrective Patches using Neural Machine Translation”. In: *arXiv preprint 1812.07170* (2018).

[173] Qinheping Hu et al. “Program Repair via Direct State Manipulation”. In: *CoRR* abs/1803.07522 (2018).

[174] Jinru Hua. “Unifying Program Repair and Program Synthesis”. PhD thesis. University of Texas at Austin, 2018.

[175] Jinru Hua et al. “Towards Practical Program Repair with On-Demand Candidate Generation”. In: *Proceedings of ICSE*. 2018.

[176] Jiajun Jiang et al. “Shaping Program Repair Space with Existing Patches and Similar Code”. In: *Proceedings of ISSTA*. 2018.

[177] René Just et al. “Comparing developer-provided to user-provided tests for fault localization and automated program repair”. In: *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2018, pp. 287–297.

[178] Xuan Bach D Le. “Overfitting in Automated Program Repair: Challenges and Solutions”. PhD thesis. Singapore Management University, 2018.

[179] Xuan Bach D. Le et al. “Overfitting in semantics-based automated program repair”. In: *Empirical Software Engineering* (2018).

[180] Junhee Lee, Seongjoon Hong, and Hakjoo Oh. “MemFix: static analysis-based repair of memory deallocation errors for C”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM. 2018, pp. 95–106.

[181] Junho Lee et al. “Automatic diagnosis and correction of logical errors for functional programming assignments”. In: *Proceedings of OOPSLA* (2018).

[182] Kui Liu. “Deep Pattern Mining for Program Repair”. PhD thesis. University of Luxembourg, 2018.

[183] Kui Liu et al. “LSRepair: Live Search of Fix Ingredients for Automated Program Repair”. In: *Proceedings of APSEC*. 2018.

[184] X. Liu and H. Zhong. “Mining Stackoverflow for Program Repair”. In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2018.

[185] Yuzhen Liu, Long Zhang, and Zhenyu Zhang. “A Survey of Test Based Automatic Program Repair”. In: *Journal of Software* (2018).

[186] Fan Long. “Automatic patch generation via learning from successful human patches”. PhD thesis. Massachusetts Institute of Technology, 2018.

[187] Sonai Mahajan et al. “Automated repair of mobile friendly problems in web pages”. In: *Proceedings of the 40th International Conference on Software Engineering*. 2018, pp. 140–150.

[188] Sonal Mahajan et al. “Automated Repair of Internationalization Presentation Failures in Web Pages Using Style Similarity Clustering and Search-Based Techniques”. In: *International Conference on Software Testing, Verification and Validation*. IEEE. 2018, pp. 215–226.

[189] Matias Martinez and Martin Monperrus. “Ultra-Large Repair Search Space with Automatically Mined Templates: the Cardumen Mode of Astor”. In: *SSBSE 2018 - 10th International Symposium on Search-Based Software Engineering*. Vol. 11036. 2018, pp. 65–86.

[190] Sergey Mechtaev. “Semantic Program Repair”. PhD thesis. National University of Singapore, 2018.

[191] Sergey Mechtaev et al. “Semantic Program Repair Using a Reference Implementation”. In: *Proceedings of the 40th International Conference on Software Engineering.* 2018.

[192] Sergey Mechtaev et al. “Test-equivalence Analysis for Automatic Patch Generation”. In: *ACM Transactions on Software Engineering and Methodology* 27.4 (2018), p. 15.

[193] Martin Monperrus et al. *Human-competitive Patches in Automatic Program Repair with Repairnator.* Tech. rep. 1810.05806. arXiv, 2018.

[194] Vinicius Paulo L Oliveira et al. “Improved representation and genetic operators for linear genetic programming for automated program repair”. In: *Empirical Software Engineering* (2018), pp. 1–27.

[195] Ripon K. Saha et al. “Bugs.jar: A Large-scale, Diverse Dataset of Real-world Java Bugs”. In: *Proceedings of the 15th International Conference on Mining Software Repositories (MSR Data Showcase).* 2018.

[196] Eddie Antonio Santos et al. “Syntax and sensibility: Using language models to detect and correct syntax errors”. In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER).* 2018, pp. 311–322.

[197] Richard Shin, Illia Polosukhin, and Dawn Song. “Towards Specification-Directed Program Repair”. In: *ICLR Workshop.* 2018.

[198] Mauricio Soto and Claire Le Goues. “Common Statement Kind Changes to Inform Automatic Program Repair”. In: *Proceedings of the 15th International Conference on Mining Software Repositories.* 2018, pp. 102–105.

[199] Eduardo Faria de Souza, Claire Le Goues, and Celso Gonçalves Camilo-Junior. “A Novel Fitness Function for Automated Program Repair Based on Source Code Checkpoints”. In: *Proceedings of the Genetic and Evolutionary Computation Conference.* GECCO ’18. 2018.

[200] Andrea Stocco, Rahulkrishna Yandrapally, and Ali Mesbah. “Visual Web Test Repair”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 2018.

[201] Shin Hwei Tan. “Design of repair operators for automated program repair”. PhD thesis. National University of Singapore, 2018.

[202] Shin Hwei Tan et al. “Repairing Crashes in Android Apps”. In: *Proceedings of the International Conference on Software Engineering.* 2018.

[203] Rijnard van Tonder and Claire Le Goues. “Static Automated Program Repair for Heap Properties”. In: *Proceedings of ICSE.* 2018.

[204] Chadi Trad et al. *CFAAR: Control Flow Alteration to Assist Repair.* Tech. rep. arXiv preprint 1808.09229, 2018.

[205] Michele Tufano et al. “An Empirical Investigation into Learning Bug-fixing Patches in the Wild via Neural Machine Translation”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering.* 2018, pp. 832–837.

[206] Michele Tufano et al. “An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation”. In: *ACM Transactions on Software Engineering and Methodology* (2018).

[207] Simon Urli et al. “How to Design a Program Repair Bot? Insights from the Repairnator Project”. In: *40th International Conference on Software Engineering, Track Software Engineering in Practice.* 2018, pp. 95–104.

[208] Marko Vasic et al. “Neural Program Repair by Jointly Learning to Localize and Repair”. In: *Proceedings of ICLR.* 2018.

[209] Kaiyuan Wang, Allison Sullivan, and Sarfraz Khurshid. “Automated model repair for Alloy”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering.* ACM. 2018, pp. 577–588.

[210] Ke Wang, Rishabh Singh, and Zhendong Su. “Search, Align, and Repair: Data-driven Feedback Generation for Introductory Programming Exercises”. In: *PLDI*. 2018, pp. 481–495.

[211] Shangwen Wang et al. “Attention Please: Consider Mockito when Evaluating Newly Released Automated Program Repair Techniques”. In: *arXiv e-prints* (2018).

[212] Ming Wen et al. “Context-Aware Patch Generation for Better Automated Program Repair”. In: *Proceedings of ICSE*. 2018.

[213] Chu-Pan Wong, Jens Meinicke, and Christian Kästner. “Beyond Testing Configurable Systems: Applying Variational Execution to Automatic Program Repair and Higher Order Mutation Testing”. In: *Proceedings of the 26th International Symposium on Foundations of Software Engineering – New Ideas Track (FSE-NIER)*. Cary, NC, 2018.

[214] Yingfei Xiong et al. “Identifying patch correctness in test-based program repair”. In: *Proceedings of ICSE*. 2018.

[215] He Ye, Matias Martinez, and Martin Monperrus. *A Comprehensive Study of Automatic Program Repair on the QuixBugs Benchmark*. Tech. rep. 1805.03454. arXiv, 2018.

[216] Zhongxing Yu et al. “Alleviating Patch Overfitting with Automatic Test Generation: A Study of Feasibility and Effectiveness for the Nopol Repair System”. In: *Empirical Software Engineering* (2018).

[217] Hao Zhong and Hong Mei. “Mining repair model for exception-related bug”. In: *Journal of Systems and Software* 141 (2018), pp. 16–31.

[218] Hao Zhong and Na Meng. “Towards reusing hints from past fixes - An exploratory study on thousands of real samples”. In: *Empirical Software Engineering* 23.5 (2018), pp. 2521–2549.

[219] Afsoon Afzal et al. “SOSRepair: Expressive Semantic Search for Real-World Program Repair”. In: *IEEE Transactions on Software Engineering* (2019).

[220] Moumita Asad, Kishan Kumar Ganguly, and Kazi Sakib. “Impact Analysis of Syntactic and Semantic Similarities on Patch Prioritization in Automated Program Repair”. In: *Proceedings of the International Conference on Software Maintenance and Evolution*. 2019.

[221] Rohan Bavishi, Hiroaki Yoshida, and Mukul R Prasad. “Phoenix: automated data-driven synthesis of repairs for static analysis violations”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM. 2019, pp. 613–624.

[222] José Pablo Cambronero et al. “Characterizing Developer Use of Automatically Generated Patches”. In: *arXiv preprint arXiv:1907.06535* (2019).

[223] Zimin Chen, Steve Kommrusch, and Martin Monperrus. *Using Sequence-to-Sequence Learning for Repairing C Vulnerabilities*. Tech. rep. 1912.02015. arXiv, 2019.

[224] Zimin Chen et al. “SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair”. In: *IEEE Transactions on Software Engineering* (2019).

[225] Xi Cheng et al. “Automatic Integer Error Repair by Proper-Type Inference”. In: *IEEE Transactions on Dependable and Secure Computing* (2019).

[226] Zhen Yu Ding et al. “Leveraging Program Invariants to Promote Population Diversity in Search-Based Automatic Program Repair”. In: *Proceedings of the Genetic Improvement Workshop*. 2019.

[227] Naji Dmeiri et al. “BugSwarm: Mining and Continuously Growing a Dataset of Reproducible Failures and Fixes”. In: *arXiv preprint arXiv:1903.06725* (2019).

[228] Thomas Durieux and Rui Abreu. “Critical Review of BugSwarm for Fault Localization and Program Repair”. In: *arXiv preprint arXiv:1905.09375* (2019).

[229] Thomas Durieux et al. “Empirical Review of Java Program Repair Tools: A Large-Scale Experiment on 2,141 Bugs and 23,551 Repair Attempts”. In: *Proceedings of FSE*. 2019.

[230] Xiang Gao, Sergey Mechtaev, and Abhik Roychoudhury. “Crash-Avoiding Program Repair”. In: *International Symposium on Software Testing and Analysis*. 2019.

[231] Ali Ghanbari. “Validation of Automatically Generated Patches: An Appetizer”. In: *arXiv preprint arXiv:1912.00117* (2019).

[232] Rongxun Guo et al. “Speedup automatic program repair using dynamic software updating: an empirical study”. In: *Proceedings of the 11th Asia-Pacific Symposium on Internetware*. 2019, pp. 1–10.

[233] Hossein Hajipour, Apratim Bhattacharya, and Mario Fritz. “SampleFix: Learning to Correct Programs by Sampling Diverse Fixes”. In: *arXiv preprint arXiv:1906.10502* (2019).

[234] Jacob Alexander Harer. “Improved neural machine translation systems for low resource correction tasks”. PhD thesis. Boston University, 2019.

[235] Yang Hu et al. “Re-factoring based Program Repair applied to Programming Assignments”. In: *Proceedings of ASE*. 2019.

[236] Zhen Huang et al. “Using Safety Properties to Generate Vulnerability Patches”. In: *Proceedings of the 40th IEEE Symposium on Security and Privacy*. 2019.

[237] Jiajun Jiang, Yingfei Xiong, and Xin Xia. “A manual inspection of Defects4J bugs and its implications for automatic program repair”. In: *Science China Information Sciences* 62.10 (2019).

[238] Besma Khaireddine, Matias Martinez, and Ali Mili. “Program Repair at Arbitrary Fault Depth”. In: *IEEE Conference on Software Testing, Validation and Verification*. 2019, pp. 465–472.

[239] Jindae Kim and Sunghun Kim. “Automatic patch generation with context-based change application”. In: *Empirical Software Engineering* (2019).

[240] Jindae Kim et al. “The effectiveness of context-based change application on automatic program repair”. In: *Empirical Software Engineering* (2019), pp. 1–36.

[241] Anil Koyuncu et al. “iFixR: bug report driven program repair”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM. 2019, pp. 314–325.

[242] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. “Automated Program Repair”. In: *Communications of the ACM* (2019).

[243] Guangpu Li et al. “DFix: automatically fixing timing bugs in distributed systems”. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2019, pp. 994–1009.

[244] Xiangyu Li, Marcelo d’Amorim, and Alessandro Orso. “Intent-Preserving Test Repair”. In: *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE. 2019, pp. 217–227.

[245] Kui Liu et al. “AVATAR: Fixing Semantic Bugs with Fix Patterns of Static Analysis Violations”. In: *Proceedings of SANER*. 2019.

[246] Kui Liu et al. “TBar: Revisiting Template-based Automated Program Repair”. In: *Proceedings of ISSTA*. 2019.

[247] Kui Liu et al. “You Cannot Fix What You Cannot Find! An Investigation of Fault Localization Bias in Benchmarking Automated Program Repair Systems”. In: *Proceedings of the 12th IEEE International Conference on Software Testing, Verification and Validation*. IEEE. 2019.

[248] Giovanni Liva et al. “Automatic repair of timestamp comparisons”. In: *IEEE Transactions on Software Engineering* (2019).

[249] Benjamin Loriot, Fernanda Madeiral, and Martin Monperrus. *Styler: Learning Formatting Conventions to Repair Checkstyle Errors*. Tech. rep. 1904.01754. arXiv, 2019.

[250] Yiling Lou et al. "Can Automated Program Repair Refine Fault Localization?" In: *arXiv preprint arXiv:1910.01270* (2019).

[251] Yiling Lou et al. "History-driven build failure fixing: how far are we?" In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM. 2019, pp. 43–54.

[252] Thibaud Lutellier et al. *ENCORE: Ensemble Learning using Convolution Neural Machine Translation for Automatic Program Repair*. 2019.

[253] Fernanda Madeiral et al. "Bears: An Extensible Java Bug Benchmark for Automatic Program Repair Studies". In: *SANER 2019 - 26th IEEE International Conference on Software Analysis, Evolution and Reengineering*. 2019.

[254] Diego Marcilio et al. "Automatically Generating Fix Suggestions in Response to Static Code Analysis Warnings". In: *19th SCAM* (2019).

[255] Alexandru Marginean et al. "SapFix: Automated End-to-End Repair at Scale". In: *International Conference on Software Engineering - Software Engineering in Practice*. 2019.

[256] Matias Martinez and Martin Monperrus. "Astor: Exploring the Design Space of Generate-and-Validate Program Repair beyond GenProg". In: *Journal of Systems and Software, Elsevier* (2019).

[257] Matias Martinez and Martin Monperrus. "Astor: Exploring the Design Space of Generate-and-Validate Program Repair beyond GenProg". In: *Journal of Systems and Software, Elsevier* (2019).

[258] Ali Mesbah et al. "DeepDelta: learning to repair compilation errors". In: *Proceedings of ESEC/FSE*. 2019.

[259] Mahmoud Mohammadi, Bill Chu, and Heather Richter Lipford. "Automated Repair of Cross-Site Scripting Vulnerabilities through Unit Testing". In: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2019, pp. 370–377.

[260] Venkatesh Theru Mohan and Ali Jannesari. "Automatic Repair and Type Binding of Undeclared Variables using Neural Networks". In: *arXiv preprint arXiv:1907.06205* (2019).

[261] Martin Monperrus. "Explainable Software Bot Contributions: Case Study of Automated Bug Fixes". In: *Proceedings of IEEE/ACM International Workshop on Bots in Software Engineering (BotSE)*. 2019.

[262] Martin Monperrus et al. "Repairnator patches programs automatically". In: *Ubiquity* 2019 (2019).

[263] Paul Muntean et al. "IntRepair: Informed Repairing of Integer Overflows". In: *IEEE Transactions on Software Engineering* (2019).

[264] Seemanta Saha, Ripon K Saha, and Mukul R Prasad. "Harnessing evolution for multi-hunk program repair". In: *Proceedings of the International Conference on Software Engineering*. 2019, pp. 13–24.

[265] Andrew Scott, Johannes Bader, and Satish Chandra. "Getafix: Learning to Fix Bugs Automatically". In: *arXiv preprint arXiv:1902.06111* (2019).

[266] August Shi et al. "iFixFlakies: a framework for automatically fixing order-dependent flaky tests". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM. 2019, pp. 545–555.

[267] Yuya Tomida et al. "Visualizing Code Genealogy: How Code is Evolutionarily Fixed in Program Repair?" In: *Proceedings of the Working Conference on Software Visualization*. 2019.

[268] Rijnard van Tonder and Claire Le Goues. “Towards s/engineer/bot: principles for program repair bots”. In: *Proceedings of the 1st International Workshop on Bots in Software Engineering*. 2019.

[269] J Tyler et al. “Trust in Automated Software Repair”. In: *First International Conference on HCI for Cybersecurity, Privacy and Trust*. 2019.

[270] Kaiyuan Wang, Allison Sullivan, and Sarfraz Khurshid. “ARepair: a repair framework for alloy”. In: *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*. 2019, pp. 103–106.

[271] Shangwen Wang et al. “How Different Is It Between Machine-Generated and Developer-Provided Patches? An Empirical Study on The Correct Patches Generated by Automated Program Repair Techniques”. In: *arXiv preprint arXiv:1906.03447* (2019).

[272] Weichao Wang et al. “LoopFix: An Approach to Automatic Repair of Buggy Loops”. In: *Journal of Systems and Software* (2019).

[273] Qi Xin and Steven P Reiss. “Better code search and reuse for better program repair”. In: *Proceedings of the 6th International Workshop on Genetic Improvement*. 2019, pp. 10–17.

[274] Qi Xin and Steven P Reiss. “Revisiting ssFix for Better Program Repair”. In: *arXiv preprint 1903.04583* (2019).

[275] Xiaoqian Xing and Katsuhisa Maruyama. “Automatic Software Merging using Automated Program Repair”. In: *International Workshop on Intelligent Bug Fixing*. 2019.

[276] Xuezheng Xu et al. “VFix: Value-Flow-Guided Precise Program Repair for Null Pointer Dereferences”. In: *Proceedings of ICSE*. 2019.

[277] Xin Yi et al. “Efficient automated repair of high floating-point errors in numerical libraries”. In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), p. 56.

[278] Xiao Liang Yu et al. “Smart Contract Repair”. In: *arXiv preprint arXiv:1912.05823* (2019).

[279] Zhongxing Yu et al. *Learning the Relation between Code Features and Code Transforms with Structured Prediction*. Tech. rep. 1907.09282. arXiv, 2019.

[280] Yuan Yuan and Wolfgang Banzhaf. “A hybrid evolutionary system for automatic software repair”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2019, pp. 1417–1425.

[281] Gene M Alarcon et al. “Would You Fix This Code for Me? Effects of Repair Source and Commenting on Trust in Code Repair”. In: *Systems* 8.1 (2020), p. 8.

[282] Aldeida Aleti and Matias Martinez. “E-APR: Mapping the Effectiveness of Automated Program Repair”. In: *arXiv preprint arXiv:2002.03968* (2020).

[283] Abdulaziz Alhefdhi et al. “Adversarial Patch Generation for Automatic Program Repair”. In: *arXiv preprint arXiv:2012.11060* (2020).

[284] Samuel Benton et al. “On the effectiveness of unified debugging: An extensive study on 16 program repair systems”. In: *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2020, pp. 907–918.

[285] Bhargav Nagaraja Bhatt and Carlo A Furia. “Automated Repair of Resource Leaks in Android Applications”. In: *arXiv preprint arXiv:2003.03201* (2020).

[286] Marcel Böhme, Charaka Geethal, and Van-Thuan Pham. “Human-In-The-Loop Automatic Program Repair”. In: *Proceedings of ICST*. 2020.

[287] Heling Cao et al. “A Survey on Automatic Bug Fixing”. In: *2020 6th International Symposium on System and Software Reliability (ISSSR)*. IEEE. 2020, pp. 122–131.

[288] Antônio Carvalho et al. “C-3PR: A Bot for Fixing Static Analysis Violations via Pull Requests”. In: (2020).

[289] Lingchao Chen and Lingming Zhang. "Fast and Precise On-the-fly Patch Validation for All". In: *arXiv preprint arXiv:2007.11449* (2020).

[290] Liushan Chen, Yu Pei, and Carlo Alberto Furia. "Contract-Based Program Repair without The Contracts: An Extended Study". In: *IEEE Transactions on Software Engineering* (2020).

[291] Zack Coker. "Automatic Repair of Framework Applications". PhD thesis. Carnegie Mellon University, 2020.

[292] V. Csuvik et al. "Utilizing Source Code Embeddings to Identify Correct Patches". In: *IEEE 2nd International Workshop on Intelligent Bug Fixing (IBF)*. 2020, pp. 18–25.

[293] Elizabeth Dinella et al. "Hoppity: learning graph transformations to detect and fix bugs in programs". In: *Proceedings of ICLR*. 2020.

[294] Yangruibo Ding et al. "Patching as Translation: the Data and the Metaphor". In: *arXiv preprint arXiv:2008.10707* (2020).

[295] Piotr Dziurzanski et al. "Empirical Analysis of 1-edit Degree Patches in Syntax-Based Automatic Program Repair". In: *IEEE Congress on Evolutionary Computation*. 2020.

[296] Khashayar Etemadi et al. *Longitudinal Analysis of the Applicability of Program Repair on Past Commits*. Tech. rep. 2007.06986. arXiv, 2020.

[297] Davide Ginelli. "Understanding and Improving Automatic Program Repair: A Study of Code-removal Patches and a New Exception-driven Fault Localization Approach". PhD thesis. Università Degli Studi Di Milano-Bicocca, 2020.

[298] Davide Ginelli et al. "A Comprehensive Study of Code-removal Patches in Automated Program Repair". In: *arXiv preprint arXiv:2012.06264* (2020).

[299] Kavi Gupta et al. "Synthesize, execute and debug: Learning to repair for neural program synthesis". In: *Advances in Neural Information Processing Systems* 33 (2020).

[300] Jingzhu He et al. "HangFix: automatically fixing software hang bugs for production cloud systems". In: *Proceedings of the 11th ACM Symposium on Cloud Computing*. 2020, pp. 344–357.

[301] Seongjoon Hong et al. "SAVER: Scalable, Precise, and Safe Memory-Error Repair". In: *Proceedings of ICSE*. 2020.

[302] Shan Huang, Xiao Zhou, and Sang Chin. "A Study of Pyramid Structure for Code Correction". In: *arXiv preprint arXiv:2001.11367* (2020).

[303] Mingyue Jiang et al. "Input Test Suites for Program Repair: A Novel Construction Method Based on Metamorphic Relations". In: *IEEE Transactions on Reliability* (2020).

[304] Anil Koyuncu. "Boosting Automated Program Repair for Adoption By Practitioners". PhD thesis. University of Luxembourg, 2020.

[305] Anil Koyuncu et al. "FixMiner: Mining Relevant Fix Patterns for Automated Program Repair". In: *Empirical Software Engineering Journal, Springer Verlag* (2020).

[306] Anil Koyuncu et al. "FlexiRepair: Transparent Program Repair with Generic Patches". In: *arXiv preprint arXiv:2011.13280* (2020).

[307] Yi Li, Shaohua Wang, and Tien N Nguyen. "DLfix: Context-based code transformation learning for automated program repair". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 2020, pp. 602–614.

[308] Bo Lin et al. "Understanding the Non-Repairability Factors of Automated Program Repair Techniques". In: *Proceedings of APSEC*. 2020.

[309] Kui Liu et al. "On the Efficiency of Test Suite based Program Repair". In: *Proceedings of ICSE*. 2020.

[310] Thibaud Lutellier et al. "CoCoNuT: Combining Context-Aware Neural Translation Models using Ensemble for Program Repair". In: *Proceedings of ISSTA*. 2020.

[311] M. Motwani et al. "Quality of Automated Program Repair on Real-World Defects". In: *IEEE Transactions on Software Engineering* (2020).

[312] Manish Motwani and Yuriy Brun. "Automatically Repairing Programs Using Both Tests and Bug Reports". In: *arXiv preprint arXiv:2011.08340* (2020).

[313] Kunihiro Noda et al. "Experience Report: How Effective is Automated Program Repair for Industrial Software?" In: *27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2020.

[314] Minxue Pan et al. "GUI-Guided Test Script Repair for Mobile Apps". In: *IEEE Transactions on Software Engineering* (2020).

[315] Georgios Sakkas et al. "Type error feedback via analytic program repair". In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2020, pp. 16–30.

[316] Haoye Tian et al. "Evaluating representation learning of code changes for predicting patch correctness in program repair". In: *Proceedings of ASE*. 2020.

[317] Yuki Ueda et al. "DevReplay: Automatic Repair with Editable Fix Pattern". In: *arXiv preprint arXiv:2005.11040* (2020).

[318] Shangwen Wang et al. "Automated Patch Correctness Assessment: How Far are We?" In: *Proceedings of ASE*. 2020.

[319] Liwei Wu et al. "GGF: A Graph-based Method for Programming Language Syntax Error Correction". In: *Proceedings of the 28th International Conference on Program Comprehension*. 2020, pp. 139–148.

[320] Tongtong Xu et al. "Restore: Retrospective fault localization enhancing automated program repair". In: *IEEE Transactions on Software Engineering* (2020).

[321] Bo Yang and Jinqiu Yang. "Exploring the Differences between Plausible and Correct Patches at Fine-Grained Level". In: *IEEE 2nd International Workshop on Intelligent Bug Fixing (IBF)*. IEEE. 2020, pp. 1–8.

[322] Michihiro Yasunaga and Percy Liang. "Graph-based, Self-Supervised Program Repair from Diagnostic Feedback". In: *arXiv preprint arXiv:2005.10636* (2020).

[323] WANG Yu et al. "Automatic Detection, Validation and Repair of Race Conditions in Interrupt-Driven Embedded Software". In: *IEEE Transactions on Software Engineering* (2020).

[324] Yuan Yuan and Wolfgang Banzhaf. "Toward Better Evolutionary Program Repair: An Integrated Approach". In: *ACM Trans. Softw. Eng. Methodol.* (2020).

[325] Yuyao Zhang et al. "Smartshield: Automatic smart contract protection made easy". In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2020, pp. 23–34.

[326] Toufique Ahmed, Noah Rose Ledesma, and Premkumar Devanbu. "SYNFIX: Automatically Fixing Syntax Errors using Compiler Diagnostics". In: *arXiv preprint arXiv:2104.14671* (2021).

[327] Miltiadis Allamanis, Henry Jackson-Flux, and Marc Brockschmidt. "Self-Supervised Bug Detection and Repair". In: *arXiv preprint arXiv:2105.12787* (2021).

[328] Ali S Alotaibi, Paul T Chiou, and William GJ Halfond. "Automated Repair of Size-Based Inaccessibility Issues in Mobile Applications". In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2021, pp. 730–742.

[329] Benoit Baudry et al. "A Software-Repair Robot Based on Continual Learning". In: *IEEE Software* 38.4 (2021), pp. 28–35.

[330] Samuel Benton et al. "Self-Boosted Automated Program Repair". In: *arXiv preprint arXiv:2104.04611* (2021).

[331] Berkay Berabi et al. "TFix: Learning to Fix Coding Errors with a Text-to-Text Transformer". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 780–791.

[332] Dirk Beyer et al. "Towards a Benchmark Set for Program Repair Based on Partial Fixes". In: *arXiv preprint arXiv:2107.08038* (2021).

[333] Diogo Campos et al. "Automatic Program Repair as Semantic Suggestions: An Empirical Study". In: *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE. 2021, pp. 217–228.

[334] Liushan Chen. "Effective automatic program repair based on state abstraction". PhD thesis. Hong Kong Polytechnic University, 2021.

[335] Zimin Chen, Steve Kommrusch, and Martin Monperrus. *Neural Transfer Learning for Repairing Security Vulnerabilities in C Code*. Tech. rep. 2104.08308. arXiv, 2021.

[336] Thanh Le-Cong et al. "Usability and Aesthetics: Better Together for Automated Repair of Web Pages". In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE. 2021, pp. 173–183.

[337] Andreea Costea et al. "HIPPODROME: Data Race Repair using Static Analysis Summaries". In: *arXiv preprint arXiv:2108.02490* (2021).

[338] Viktor Csuvik et al. "Exploring Plausible Patches Using Source Code Embeddings in JavaScript". In: *arXiv preprint arXiv:2103.16846* (2021).

[339] Khashayar Etemadi et al. *Sorald: Automatic Patch Suggestions for SonarQube Static Analysis Violations*. Tech. rep. 2103.12033. arXiv, 2021.

[340] Xiang Gao. "Overfitting in Program Repair and Synthesis". PhD thesis. National University of Singapore, 2021.

[341] Xiang Gao et al. "Beyond Tests: Program Vulnerability Repair via Crash Constraint Extraction". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30.2 (2021), pp. 1–27.

[342] Jingzhu He, Ting Dai, and Xiaohui Gu. "TFix+: Self-configuring Hybrid Timeout Bug Fixing for Cloud Systems". In: *arXiv preprint arXiv:2110.04101* (2021).

[343] Jordan Henkel et al. "Shipwright: A Human-in-the-Loop System for Dockerfile Repair". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 1148–1160.

[344] Nan Jiang, Thibaud Lutellier, and Lin Tan. "CURE: Code-Aware Neural Machine Translation for Automatic Program Repair". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 1161–1173.

[345] Maria Kechagia et al. "Evaluating Automatic Program Repair Capabilities to Repair API Misuses". In: *IEEE Transactions on Software Engineering* (2021).

[346] Serkan Kirbas et al. "On the introduction of automatic program repair in Bloomberg". In: *IEEE Software* 38.4 (2021), pp. 43–51.

[347] William Klieber et al. "Automated Code Repair to Ensure Spatial Memory Safety". In: *2021 IEEE/ACM International Workshop on Automated Program Repair (APR)*. IEEE. 2021, pp. 23–30.

[348] Roland Kretschmer, Djamel Eddine Khelladi, and Alexander Egyed. "Transforming abstract to concrete repairs with a generative approach of repair values". In: *Journal of Systems and Software* 175 (2021), p. 110889.

[349] Xuan-Bach D Le and Quang Loc Le. "ReFixar: Multi-version Reasoning for Automated Repair of Regression Errors". In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE. 2021, pp. 162–172.

[350] Kui Liu et al. "A critical review on the evaluation of automated program repair systems". In: *Journal of Systems and Software* 171 (2021), p. 110817.

[351] Yiling Lou et al. “How Does Regression Test Selection Affect Program Repair? An Extensive Study on 2 Million Patches”. In: *arXiv preprint arXiv:2105.07311* (2021).

[352] Yunlong Lu, Na Meng, and Wenxin Li. “FAPR: Fast and Accurate Program Repair for Introductory Programming Courses”. In: *arXiv preprint arXiv:2107.06550* (2021).

[353] Ibrahim Mesecan et al. “CRNRepair: Automated Program Repair of Chemical Reaction Networks”. In: *2021 IEEE/ACM International Workshop on Genetic Improvement (GI)*. IEEE. 2021, pp. 23–30.

[354] Marjane Namavar, Noor Nashid, and Ali Mesbah. “A Controlled Experiment of Different Code Representations for Learning-Based Bug Repair”. In: *arXiv preprint 2110.14081* (2021).

[355] Tai D Nguyen, Long H Pham, and Jun Sun. “SGUARD: Towards Fixing Vulnerable Smart Contracts Automatically”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1215–1229.

[356] Thanh-Toan Nguyen et al. “Automated Repair of Heap-Manipulating Programs using Deductive Synthesis”. In: *Proceedings of VMCAI*. 2021.

[357] Amirfarhad Nilizadeh et al. “Exploring true test overfitting in dynamic automated program repair using formal methods”. In: *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE. 2021, pp. 229–240.

[358] Amirfarhad Nilizadeh et al. “More Reliable Test Suites for Dynamic APR by using Counterexamples”. In: *2021 IEEE 32th International Symposium on Software Reliability Engineering (ISSRE)(In Press). IEEE*. 2021.

[359] Yannic Noller et al. “How to trust auto-generated code patches? A developer survey and empirical assessment of existing program repair tools”. In: *arXiv preprint arXiv:2108.13064* (2021).

[360] Vesna Nowack et al. “Expanding Fix Patterns to Enable Automatic Program Repair”. In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE. 2021, pp. 12–23.

[361] Eduard Pinconschi, Rui Abreu, and Pedro Adão. “A Comparative Study of Automatic Program Repair Techniques for Security Vulnerabilities”. In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE. 2021, pp. 196–207.

[362] Kai Presler-Marshall, Sarah Heckman, and Kathryn T Stolee. “SQLRepair: Identifying and Repairing Mistakes in Student-Authored SQL Queries”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE. 2021, pp. 199–210.

[363] Yihao Qin et al. “On the Impact of Flaky Tests in Automated Program Repair”. In: *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2021, pp. 295–306.

[364] Kia Rahmani et al. “Repairing serializability bugs in distributed database programs via automated schema refactoring”. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 2021, pp. 32–47.

[365] Joseph Renzullo, Westley Weimer, and Stephanie Forrest. “Multiplicative Weights Algorithms for Parallel Automated Software Repair”. In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2021, pp. 984–993.

[366] Michael Rodler et al. “EVMPatch: Timely and Automated Patching of Ethereum Smart Contracts”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 1289–1306.

[367] Ridwan Shariffdeen et al. “Concolic program repair”. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 2021, pp. 390–405.

[368] Ben Tang et al. "GrasP: Graph-to-Sequence Learning for Automated Program Repair". In: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE. 2021, pp. 819–828.

[369] Yu Tang et al. "Grammar-Based Patches Generation for Automated Program Repair". In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 2021, pp. 1300–1305.

[370] Christof Ferreira Torres, Hugo Jonker, and Radu State. "Elysium: Automagically Healing Vulnerable Smart Contracts Using Context-Aware Patching". In: *arXiv preprint arXiv:2108.10071* (2021).

[371] Leonardo Trujillo, Omar M Villanueva, and Daniel Eduardo Hernandez. "A novel approach for search-based program repair". In: *IEEE Software* 38.4 (2021), pp. 36–42.

[372] Chu-Pan Wong et al. "VarFix: balancing edit expressiveness and search effectiveness in automated program repair". In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2021, pp. 354–366.

[373] Deheng Yang et al. "Is the Ground Truth Really Accurate? Dataset Purification for Automated Program Repair". In: *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2021, pp. 96–107.

[374] Deheng Yang et al. "Where were the repair ingredients for Defects4j bugs?" In: *Empirical Software Engineering* 26.6 (2021), pp. 1–33.

[375] Michihiro Yasunaga and Percy Liang. "Break-It-Fix-It: Unsupervised Learning for Program Repair". In: *arXiv preprint arXiv:2106.06600* (2021).

[376] He Ye, Matias Martinez, and Martin Monperrus. "Automated Patch Assessment for Program Repair at Scale". In: *Empirical Software Engineering* (2021).

[377] He Ye, Matias Martinez, and Martin Monperrus. "Neural Program Repair with Execution-based Backpropagation". In: *arXiv preprint arXiv:2105.04123* (2021).

[378] He Ye et al. "Automated Classification of Overfitting Patches with Statically Extracted Code Features". In: *IEEE Transactions on Software Engineering* (2021).

[379] Jingtang Zhang et al. "Revisiting Test Cases to Boost Generate-and-Validate Program Repair". In: *Proceedings of ICSME*. 2021.

[380] Qihao Zhu et al. "A Syntax-Guided Edit Decoder for Neural Program Repair". In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2021.

[381] Hammad Ahmad, Yu Huang, and Westley Weimer. "CirFix: Automatically Repairing Defects in Hardware Design Code". In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2022.

[382] Liushan Chen et al. "Program Repair with Repeated Learning". In: *IEEE Transactions on Software Engineering* (2022).

[383] Aidan Connor et al. "Can We Automatically Fix Bugs by Learning Edit Operations?" In: *Proceedings of the 29th IEEE International Conference on Software Analysis, Evolution and Reengineering*. 2022.

[384] Khashayar Etemadi et al. "Estimating the potential of program repair search spaces with commit analysis". In: *Journal of Systems and Software* (2022), p. 111263.

[385] Davide Ginelli et al. "Exception-Driven Fault Localization for Automated Program Repair". In: *arXiv preprint arXiv:2201.00736* (2022).

[386] Yaojie Hu et al. "Fix Bugs with Transformer through a Neural-Symbolic Edit Grammar". In: *arXiv preprint arXiv:2204.06643* (2022).

[387] Sungmin Kang and Shin Yoo. "GLAD: Neural Predicate Synthesis to Repair Omission Faults". In: *arXiv preprint arXiv:2204.06771* (2022).

[388] Hyungsub Kim et al. "PGPATCH: Policy-Guided Logic Bug Patching for Robotic Vehicles". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2022, pp. 1538–1538.

[389] Junhee Lee, Seongjoon Hong, and Hakjoo Oh. "NPEX: Repairing Java Null Pointer Exceptions without Tests". In: *IEEE International Conference on Software Engineering.* 2022.

[390] Chao Ni et al. "Defect Identification, Categorization, and Repair: Better Together". In: *arXiv preprint arXiv:2204.04856* (2022).

[391] Yannic Noller et al. "Trust Enhancement Issues in Program Repair". In: *Proceedings of ICSE.* 2022.

[392] Quang-Ngoc Phung, Misoo Kim, and Eunseok Lee. "Identifying Incorrect Patches in Program Repair Based on Meaning of Source Code". In: *IEEE Access* 10 (2022), pp. 12012–12030.

[393] Pemma Reiter et al. "Automatically Mitigating Vulnerabilities in x86 Binary Programs via Partially Recompilable Decompilation". In: *arXiv preprint arXiv:2202.12336* (2022).

[394] Lei Shi et al. "Automatic Repair for Network Programs". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer. 2022, pp. 353–372.

[395] Emily Rowan Winter et al. "Let's Talk With Developers, Not About Developers: A Review of Automatic Program Repair Research". In: *IEEE Transactions on Software Engineering* (2022).

[396] Xuezheng Xu, Xudong Wang, and Jingling Xue. "M3V: Multi-modal Multi-view Context Embedding for Repair Operator Prediction". In: *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE. 2022, pp. 266–277.

[397] Jooyong Yi and Elkhan Ismayilzada. "Speeding up constraint-based program repair using a search-based technique". In: *Information and Software Technology* (2022), p. 106865.

[398] Aleksandr Zakharchenko. "A practical approach to automated software correctness enhancement". PhD thesis. New Jersey Institute of Technology, 2022.

[399] Quanjun Zhang et al. "Program Repair: Automated vs. Manual". In: *arXiv preprint arXiv:2203.05166* (2022).

[400] Ying Zhang et al. "Example-Based Vulnerability Detection and Repair in Java Code". In: *arXiv preprint arXiv:2203.09009* (2022).

[401] Wenkang Zhong et al. "Neural Program Repair: Systems, Challenges and Solutions". In: *arXiv preprint arXiv:2202.10868* (2022).