DOI: 10.30750/ijarst.080404

Advances in Automated Software Quality Assurance, Defect Detection, and Codebase Reliability Engineering

Ijeoma Mbonu[1*], Emmanuel Chukwuma Okeke[2], Precious Nnamdi Ibe[3], Kingsley Chinazaekpere Ndupu[4], Uchechi Unamma[5], Oluchi Aneke[6]

[1]Adeleke University, Nigeria
[2] IDEX Corporation, USA.
[3] University of Port Harcourt, Port Harcourt, Nigeria
[4] Kennesaw State University, Georgia, USA ORCID: https://orcid.org/0009-0004-1981-479X
[5] Federal Ministry of Mines and Steel Development, Abuja, Nigeria
[6] Deloitte and Touche Nigeria, Nigeria

Abstract

The period to date has marked a turning point for automated software quality assurance. Large language models trained on source code moved from research curiosity to developer tooling, neural program repair systems began to outperform template based approaches on standard benchmarks, and deep learning vulnerability detectors were subjected to the first rigorous, realistic evaluations that exposed their limits. At the same time, a series of high profile incidents, from the SolarWinds compromise to the Log4Shell vulnerability, shifted the focus of reliability engineering from individual services to the integrity of entire software supply chains. Using a structured narrative review of 200 sources published to date, this paper synthesises the principal advances of the year across four connected areas: learned representations of code, automated defect detection and prediction, automated program repair, and test quality, together with the broader discipline of codebase reliability engineering. It also connects these advances to adjacent research streams on reliability engineering in cyber-physical systems, anomaly and fraud detection, continuous security validation, governance and audit automation, and platform engineering, drawing on one hundred related studies from a consolidated peer literature published to date. For each area we summarise the key techniques, the empirical evidence that supported or challenged them, and the industrial adoption that has begun. We then propose the Integrated Continuous Quality Assurance (ICQA) model, a five-layer conceptual model linking data and evidence, detection, verification and repair, delivery and runtime reliability, and governance through an upward assurance flow and a downward feedback loop, and derive five testable propositions. We argue that the defining feature of this period was convergence: machine learning, static analysis, testing, and operations practice began to be treated as parts of a single quality pipeline rather than as separate disciplines. We discuss implications for research, practice, and policy, state the limitations of the review, and close by identifying future research on the open problems that remain unresolved, including dataset realism, evaluation validity, the security of machine generated code, and the measurement of reliability at the scale of whole ecosystems.

Keywords: software quality assurance; defect prediction; vulnerability detection; automated program repair; mutation testing; fuzzing; flaky tests; site reliability engineering; software supply chain security; large language models for code

INTRODUCTION

Software quality assurance has always been a contest between the growth of code and the growth of the means to check it. Modern organisations maintain repositories measured in hundreds of millions of lines, merge thousands of changes per day, and depend on dense graphs of open source packages that no single team fully understands. Manual inspection and hand written tests remain essential, but they cannot keep pace on their own (Mbonu et al., 2021b; Agboada et al., 2018). The research agenda of automated quality assurance therefore asks how far detection, diagnosis, and correction of defects can be delegated to tools, and how the reliability of the resulting systems can be engineered and measured (Gideon et al., 2019; Asibor & Isong, 2019).

The year 2021 is a useful lens through which to examine this agenda because several long running

lines of work matured at once. Pre-trained transformer models of code, introduced in 2020 and building on a decade of work on the statistical naturalness of software (Hindle et al., 2012; Allamanis et al., 2018; Hellendoorn & Devanbu, 2017), were extended with structural information and generative capability (Guo et al., 2021; Wang et al., 2021; Ahmad et al., 2021), and the release of Codex and the GitHub Copilot technical preview demonstrated that models trained on public code could synthesise plausible functions from natural language descriptions (M. Chen et al., 2021). In parallel, the defect detection community published sobering re-evaluations showing that reported accuracies of deep vulnerability detectors collapsed on realistic data (Chakraborty et al., 2021), prompting new datasets (Zheng et al., 2021; Bhandari et al., 2021) and new model designs (Li et al., 2021). Program repair research reported substantial benchmark gains from neural architectures (Jiang et al., 2021; Zhu et al., 2021; Berabi et al., 2021), and self-supervised training schemes that learned to fix code without labelled pairs (Yasunaga & Liang, 2021). Industrial studies of mutation testing (Petrović et al., 2021a; Petrović et al., 2021b), fuzzing benchmarks (Metzman et al., 2021), and flaky test prediction (Alshammari et al., 2021) provided evidence on what actually works at scale.

Outside the research literature, 2021 was the year in which software supply chain integrity became a policy issue. United States Executive Order 14028 in May directed federal agencies toward secure development practices and software bills of materials (Executive Office of the President, 2021; National Telecommunications and Information Administration, 2021); Google proposed the SLSA framework for build provenance (Lewandowski & Lodato, 2021); and the Sigstore project began offering free code signing for open source (Linux Foundation, 2021). The disclosure of CVE-2021-44228 (Log4Shell) in December illustrated, in the most practical way possible, why these measures mattered (National Institute of Standards and Technology, 2021).

Despite this activity, the literature remains fragmented. Work on code models, defect detection, repair, testing, and operations is published in different communities and evaluated with different measures, and related ideas developed in reliability, security, and governance research are rarely connected to software quality work. This paper addresses that gap with three objectives: to synthesise the principal advances of 2021 across automated quality assurance, defect detection, and codebase reliability engineering; to propose a conceptual model that integrates them into a single continuous assurance pipeline; and to draw out implications, limitations, and directions for future research.

The paper is organised as follows. Section 2 describes the research method. Section 3 presents the results and discussion: a literature synthesis covering learned representations of code, defect detection and prediction, program repair, testing, reliability engineering, and adjacent research streams, followed by the conceptual model, a discussion of cross-cutting issues, and implications for research, practice, and policy. Section 4 concludes and states the limitations of the review and directions for future research.

RESEARCH METHOD

This study adopts a structured narrative review design, combining the transparency of a documented search and selection protocol with the interpretive synthesis needed to connect work from several research communities. We use the term *automated software quality assurance* to denote techniques that reduce the human effort needed to establish that software behaves as intended, including static analysis, dynamic testing, learned defect models, and automated repair. *Defect detection* refers to identifying faults or vulnerabilities in code, either before or after they manifest as failures. *Codebase reliability engineering* refers to the practices and tooling that keep a large, evolving codebase and the services built from it dependable over time, spanning continuous integration, release engineering, observability, incident response, and dependency management (Dosunmu & Ogundele, 2021; Anyanwu, 2019).

Two complementary source pools were used. The first, the *core literature*, was assembled by searching the proceedings of the principal software engineering venues (ICSE, ESEC/FSE, ASE, ISSTA, MSR,

and IEEE Transactions on Software Engineering), machine learning venues that hosted significant code related work (ICML, ICLR, NeurIPS, NAACL, EMNLP), and security venues, supplemented by primary industrial and government sources such as engineering reports, standards, and policy documents. Search terms combined the review's three themes (for example, defect prediction, vulnerability detection, program repair, mutation testing, fuzzing, flaky tests, site reliability, and software supply chain) with technique terms such as pre-trained model, graph neural network, and continuous integration. Backward snowballing from key 2021 papers identified the foundational work they build on. The second pool, the *consolidated peer literature***,** was a curated, deduplicated reference list of applied studies in cybersecurity, cloud and IT service management, reliability and maintenance engineering, analytics, and technology governance, screened for relevance to software quality and reliability.

Sources were included if they were published in or before 2021; addressed software quality assurance, defect or vulnerability detection, repair, testing, reliability, or security assurance directly, or addressed an analogous problem (such as anomaly detection, inspection, predictive maintenance, or compliance automation) in a way that informs software quality practice; and were peer reviewed, published by a recognised standards or government body, or were primary industrial reports from organisations operating the systems described. Sources were excluded if they were published after 2021, if their bibliographic record was incomplete, or if their subject matter had no reasonable connection to software or systems quality, such as agricultural, clinical, or educational studies without a technology assurance component.

Screening proceeded in two stages: title and abstract review against the inclusion criteria, followed by full-record review to confirm relevance and publication date. From the consolidated peer literature, 256 entries met the date criterion; one was excluded for an incomplete record, and the remainder were ranked by topical relevance, with the 100 most relevant retained. From the core literature, 100 sources were retained after screening. The final corpus therefore comprises 200 sources: 100 core and 100 consolidated. Within the core corpus, priority was given to 2021 publications that introduced a new technique, dataset, or evaluation, or that reported large-scale industrial evidence, while earlier works were retained where needed to explain what was new.

Core sources were coded by theme (code representation, detection, repair, testing, reliability), by type of contribution (technique, dataset or benchmark, empirical evaluation, industrial report, or policy), and by the evaluation evidence offered. Consolidated sources were coded by the software quality concept they parallel, for example inspection as an analogue of static analysis or fraud analytics as an analogue of defect prediction, and are treated as related or semi-related evidence rather than as primary software engineering results. The coded themes were then integrated into the conceptual model presented in Section 3. Table 1 lists the representative 2021 contributions that anchor the synthesis.

RESULTS AND DISCUSSION

This section reports the results of the review and discusses them. It first presents the literature synthesis in six strands. The first five follow the quality pipeline from how code is represented, through how defects are detected and repaired and how tests are generated and assessed, to how reliability is engineered in production. The sixth connects these strands to adjacent research streams that develop the same ideas in other engineering domains. The findings are then integrated in a conceptual model, followed by a discussion of cross-cutting issues and the implications for research, practice, and policy.

CodeBERT, published in late 2020, showed that the masked language modelling recipe behind BERT could be applied to paired natural language and code to produce a general purpose encoder (Feng et al., 2020). Its limitation was that it treated programs as flat token sequences, ignoring the semantic structure that compilers and analysers rely on; CuBERT had made a similar point by evaluating contextual embeddings on a suite of code understanding tasks (Kanade et al., 2020). GraphCodeBERT, presented at ICLR 2021, addressed this by incorporating data

Table 1: Representative advances of 2021 discussed in this review

Area	*Contribution (venue)*	*Principal advance*
Code representation	GraphCodeBERT, ICLR (Guo et al., 2021)	Pre-training with data flow structure in addition to token sequences
Code representation	CodeT5, EMNLP; PLBART, NAACL (Wang et al., 2021; Ahmad et al., 2021)	Encoder-decoder pre-training enabling both understanding and generation tasks
Code representation	Codex and HumanEval (M. Chen et al., 2021)	Large scale generative model of code with execution based functional evaluation
Benchmarking	CodeXGLUE, NeurIPS D&B (Lu et al., 2021)	Unified benchmark suite spanning ten code intelligence tasks
Vulnerability detection	ReVeal, TSE (Chakraborty et al., 2021)	Realistic re-evaluation exposing data duplication and imbalance problems
Vulnerability detection	IVDetect, ESEC/FSE (Li et al., 2021)	Graph based detection with fine-grained, statement level interpretation
Datasets	D2A, ICSE-SEIP; CVEfixes, PROMISE (Zheng et al., 2021; Bhandari et al., 2021)	Labelled vulnerability data from differential static analysis and CVE fix commits
Defect prediction	JITLine, MSR (Pornprasit & Tantithamthavorn, 2021)	Simple, explainable just-in-time prediction with line level localisation
Program repair	CURE, ICSE; Recoder, ESEC/FSE (Jiang et al., 2021; Zhu et al., 2021)	Code-aware neural translation and syntax guided edit decoding
Program repair	TFix and Break-It-Fix-It, ICML (Berabi et al., 2021; Yasunaga & Liang, 2021)	Pre-trained text-to-text repair and self-supervised learning without labelled pairs
Testing	Mutation testing at Google, ICSE and TSE (Petrović et al., 2021a; Petrović et al., 2021b)	Longitudinal evidence that surfaced mutants improve developer test practice
Testing	FlakeFlagger, ICSE (Alshammari et al., 2021)	Feature based prediction of flaky tests without repeated reruns
Fuzzing	FuzzBench, ESEC/FSE (Metzman et al., 2021)	Open, reproducible fuzzer evaluation service
Supply chain	EO 14028, SBOM minimum elements, SLSA, Sigstore (Executive Office of the President, 2021; National Telecommunications and Information Administration, 2021; Lewandowski & Lodato, 2021; Linux Foundation, 2021)	Policy and tooling for provenance, signing, and component transparency
Reliability practice	DORA State of DevOps 2021 (Smith et al., 2021)	Reliability adopted as a fifth delivery performance metric

flow edges into pre-training, using objectives that asked the model to predict where a variable's value comes from and to align code tokens with data flow nodes (Guo et al., 2021). The model improved results on code search, clone detection, translation, and refinement, and its design signalled a broader shift: representations useful for quality assurance needed to encode how values move through a program, not merely which tokens appear.

Two encoder-decoder models published in 2021 extended this line toward generation. PLBART adapted denoising sequence-to-sequence pre-training to a large corpus of Java and Python functions together with developer forum text (Ahmad et al., 2021). CodeT5 combined a T5 style architecture with an identifier aware objective that taught the model to recognise and recover identifiers, the parts of code that carry most developer intent (Wang et al., 2021). Because both models could emit code as well as classify it, they became natural backbones for repair, summarisation, and test generation, which were previously served by task specific architectures.

The most visible development of 2021 was the arrival of large generative models of code. M. Chen et al. (2021) described Codex, a GPT family model fine-tuned on public GitHub code, and introduced HumanEval, a set of 164 hand written programming problems scored by executing unit tests rather than

by textual similarity. The best Codex model solved 28.8 per cent of problems with a single sample and 70.2 per cent when 100 samples were drawn and any passing sample was counted. Austin et al. (2021) reported comparable findings with the MBPP benchmark and showed that performance scaled smoothly with model size. GitHub released Copilot, built on Codex, as a technical preview in June 2021, following earlier transformer-based completion in IntelliCode Compose (Svyatkovskiy et al., 2020). The APPS benchmark (Hendrycks et al., 2021) and the CodeNet dataset (Puri et al., 2021), both released in 2021, broadened execution-based evaluation to competitive programming problems and to millions of submissions across many languages.

For quality assurance, the methodological contribution of this work was as important as the models. The ***pass@k*** metric, which estimates the probability that at least one of k sampled programs passes the tests, replaced token overlap metrics such as BLEU that had been shown to correlate poorly with correctness. Execution based evaluation tied the measurement of generative models directly to the core concern of testing: does the code do what it should? The same period also surfaced the risks. Pearce et al. (2021) generated programs for 89 security relevant scenarios and found that roughly 40 per cent of the Copilot outputs they examined contained weaknesses from the MITRE CWE Top 25 list. Its deployment also raised questions about how organisations should govern data, confidentiality, and automated decisions around such tools, including privacy by design and legal and ethical risk modelling for data protection (Ominyi, 2021; Mbonu et al., 2018). Generative assistance therefore created a new obligation for quality tooling: code written with machine help needed the same, or greater, scrutiny as code written without it.

CodeXGLUE gathered fourteen datasets across ten tasks, including defect detection, clone detection, code repair, and code translation, into a single benchmark with baseline systems and a public leaderboard (Lu et al., 2021). Its defect detection task was derived from the Devign dataset of C functions (Zhou et al., 2019). Standardisation of this kind accelerated comparison between models, but, as discussed below for vulnerability detection, it also concentrated attention on datasets whose realism was later questioned. Work outside software engineering on data quality in health information systems (Ekemezie et al., 2018; Ilodigwe & Adesemoye, 2021), integrated laboratory information systems for detection accuracy (Abdulrazaq et al., 2019), and diagnostic quality systems (Yusuf & Abdulrazaq, 2018) addresses the same underlying problem: a learned quality model is only as trustworthy as the data and evaluation protocol behind it.

Rule based static analysis remained the workhorse of industrial defect detection in 2021. Semantic query engines such as CodeQL, integrated into GitHub code scanning since late 2020, allowed security teams to express vulnerability patterns as queries over a relational representation of the program and run them on every pull request. Lightweight pattern matchers gained popularity because developers could write rules in a syntax resembling the code being checked. Industrial reports from Coverity, Google, and Meta (Bessey et al., 2010; Calcagno et al., 2015; Sadowski et al., 2018; Distefano et al., 2019) established that the value of an analyser depended less on its theoretical precision than on how its results reached developers: reports surfaced during code review, with low false positive rates and clear explanations, were acted upon, whereas batch reports were largely ignored, a pattern consistent with developer studies of why analysis tools go unused (Johnson et al., 2013) and with evidence that static bug detectors find only a fraction of real bugs (Habib & Pradel, 2018). Reviews published by 2021 described the growing role of artificial intelligence in secure software testing and automated regression control (Mbonu et al., 2021b), and in malware classification for cloud-native microservices (Idika et al., 2021). The same analysis mindset spread from application code to infrastructure definitions, with governance approaches for infrastructure as code and blockchain-backed audit trails for cloud configuration management treating configuration as a first-class source of defects (Mbonu et al., 2020b; Oshoba et al., 2020).

Static analysis also became a source of training data. The D2A dataset, released by IBM researchers

at ICSE-SEIP 2021, ran the Infer analyser on consecutive versions of open source C projects and labelled issues that disappeared after a bug fixing commit as likely true positives (Zheng et al., 2021). This differential labelling produced over a million labelled examples at a fraction of the cost of manual annotation and created a bridge between symbolic and statistical approaches: the analyser proposed candidates, and the model learned which candidates were real.

Between 2018 and 2020, a sequence of papers reported high accuracy for neural vulnerability detectors operating on token sequences, abstract syntax trees, or graph representations of code (Li et al., 2018a; Russell et al., 2018; Zhou et al., 2019; Li et al., 2018b). In 2021, Chakraborty et al. (2021) tested these claims under realistic conditions. They assembled a dataset from real vulnerability fixes in the Chromium and Debian projects and found that state of the art models lost most of their reported performance, with precision and F1 scores falling sharply when moved from the curated or synthetic datasets on which they were developed. Their analysis identified three causes: duplicated samples that leaked between training and test sets, unrealistic class balance that ignored the rarity of vulnerable code, and models that learned superficial lexical features such as particular API names rather than the semantics of a flaw. The duplication problem has an analogue in national surveillance and reporting networks, where records arriving from many laboratories must be reconciled before analysis (Yusuf et al., 2020).

The authors proposed ReVeal, which combined a gated graph neural network over code property graphs (Yamaguchi et al., 2014) with re-sampling to address imbalance and a representation learning stage designed to separate vulnerable from non-vulnerable examples more clearly. ReVeal improved on prior models under realistic evaluation, but the larger contribution was the method of evaluation itself. Following this work, reviewers have begun to expect deduplication, realistic class ratios, and cross-project testing as standard. Similar concerns about imbalance, feature selection, and robustness recur in machine learning for intrusion detection, malware classification in cloud-native microservices, and anti-money laundering detection (Dosunmu & Ogundele, 2020; Idika et al., 2021; Atakpa, 2021).

A second theme was interpretability. A function level prediction that code is vulnerable is of limited use to a developer facing a function of several hundred lines. IVDetect, presented at ESEC/FSE 2021, represented each function as a program dependence graph enriched with contextual information and applied a graph explanation technique (Ying et al., 2019) to identify the sub-graph of statements most responsible for a positive prediction (Li et al., 2021). Reporting the likely vulnerable statements, rather than only a label, aligned learned detection with the way static analysis findings are consumed in code review.

Dataset construction advanced in parallel. Big-Vul, published in 2020, linked CVE entries to fixing commits in C and C++ projects (Fan et al., 2020). CVEfixes, released in 2021, automated the collection of vulnerability fixes from the National Vulnerability Database across many languages and packaged them with metadata at the repository, commit, file, and method levels, with a tool that allowed the dataset to be regenerated as new CVEs were published (Bhandari et al., 2021). Reproducible, refreshable datasets addressed one of the most persistent weaknesses of the field, which was that models were often trained and tested on static snapshots that quickly became unrepresentative.

Defect prediction research, which estimates the likelihood that a file, module, or change contains a defect, moved further toward fine grained and actionable output in 2021. Just-in-time prediction scores individual commits at the time they are made, when the context is fresh in the developer's mind (Kamei et al., 2013). Deep models of code changes such as DeepJIT and CC2Vec (Hoang et al., 2019; Hoang et al., 2020) and learned semantic features (Wang et al., 2016) had raised expectations, but a 2021 replication found that a simple logistic regression over change features could rival them (Zeng et al., 2021), echoing long-standing concerns about validation practice in defect prediction (Tantithamthavorn et al., 2017). JITLine, presented at MSR 2021, showed that a comparatively simple

model, using bag-of-tokens features of the change combined with established change metrics and an explainable classifier, could match or exceed more complex deep learning approaches while running far faster (Pornprasit & Tantithamthavorn, 2021). It also ranked the lines within a risky commit by their contribution to the prediction, helping reviewers decide where to look first, a capability practitioners reported as one of their main goals for defect prediction in a survey published the same year (Jiarpakdee et al., 2021). This finding echoed earlier work on line level prediction using model agnostic explanations (Wattanakriengkrai et al., 2020) and reinforced a theme that recurs throughout this review: in practical settings, the cost, speed, and explainability of a technique often matter as much as its headline accuracy. The same preference for interpretable early warning appears in predictive maintenance for e-commerce systems, interpretable failure prediction for distributed assets, and condition monitoring in critical power infrastructure (Mayo et al., 2021; Komi & Adamolekun, 2021; Adaramola et al., 2018).

Code review is a primary defect detection mechanism in most organisations, and 2021 saw serious attempts to automate parts of it. Tufano et al. (2021) trained transformer models on review histories to perform two tasks: revising submitted code as a reviewer would request, and generating the reviewer's comment itself. The models produced exact matches with real reviewer behaviour in a meaningful minority of cases. Although far from replacing human review, the work defines tasks and datasets on which future pre-trained review models can build, and it framed automated review as an assistant that handles routine requests so that people can concentrate on design and logic. Systematic reviews of standardised contract review workflows (Uchechi et al., 2021) and governance frameworks for platform management in regulated environments (Badmus et al., 2019b) show the same structured-review pattern spreading beyond source code.

Automated program repair (APR) seeks to produce patches that make a failing program pass its tests or that remove a detected defect. Surveys of the field (Monperrus, 2018; Le Goues et al., 2019) describe how earlier generations of APR relied on search over mutation operators or on templates mined from historical fixes, and industry systems such as Getafix and SapFix had demonstrated the value of template based repair at Meta (Bader et al., 2019; Marginean et al., 2019). Neural approaches framed repair as translation from buggy to fixed code (Lutellier et al., 2020; Z. Chen et al., 2021), and graph-based models learned to transform programs to detect and fix bugs (Dinella et al., 2020). In 2021 two systems advanced this framing substantially.

CURE, presented at ICSE 2021, pre-trained a programming language model on a large code corpus, added a code-aware beam search that discarded candidates referring to identifiers not in scope, and used subword tokenisation to reduce the vocabulary problem posed by arbitrary identifier names (Jiang et al., 2021). It repaired more bugs on the Defects4J and QuickBugs benchmarks than previous neural approaches. Recoder, presented at ESEC/FSE 2021, replaced token generation with a syntax guided edit decoder that produced modifications to the abstract syntax tree, guaranteeing syntactically valid patches and introducing placeholder generation to handle project specific identifiers (Zhu et al., 2021). Recoder was reported as the first learning based technique to outperform the best traditional template based techniques on Defects4J v1.2 (Just, Jalali, & Ernst, 2014).

TFix, presented at ICML 2021, fine-tuned a large pre-trained text-to-text model to fix errors flagged by a JavaScript linter, providing the error message and surrounding code as input (Berabi et al., 2021). Because the model had absorbed general knowledge of code during pre-training, a single model handled more than fifty error types, and it fixed a substantially higher fraction of errors than prior learned approaches. TFix illustrates a pattern likely to shape future work: pairing an analyser that identifies a problem with a large model that proposes the remedy.

A persistent obstacle for learned repair was the scarcity of paired buggy and fixed examples. Break-It-Fix-It (BIFI), also presented at ICML 2021, addressed this by training a fixer and a breaker together (Yasunaga & Liang, 2021). A critic, which

can be as simple as a compiler or parser, determines whether code is correct. The fixer is applied to real broken code, and its outputs that the critic accepts become new training pairs; the breaker learns to produce realistic errors from correct code, generating additional pairs. Iterating the cycle produced a fixer that outperformed models trained on synthetically perturbed code for syntax repair in Python and compilation repair in C. DeepDebug from Microsoft applied related ideas, combining stack traces, back-translation, and code skeletons to fix Python bugs (Drain et al., 2021). BugLab, presented at NeurIPS 2021, trained a bug detector and a bug-inserting selector against each other in the same self-supervised spirit (Allamanis et al., 2021). Together, these systems showed that the execution environment itself could serve as a supervisor, reducing dependence on manual labelling.

Benchmark gains in repair carried a known caveat: a patch that passes the available tests may still be wrong, because test suites seldom specify behaviour completely. Throughout 2021, studies of overfitting patches and automated patch correctness assessment emphasised that plausible patches must be distinguished from correct ones (Wang et al., 2020; Tian et al., 2020; Ye et al., 2021), typically by manual inspection against the developer's fix. Repair systems were therefore increasingly positioned as suggestion engines whose output enters normal code review, rather than as autonomous agents that commit changes. This positioning connected repair to the automated code review work described above, and to the wider literature on near-miss reporting and organisational learning, which shows how errors caught by people feed back into safer processes (Arumosoye & Obriki, 2019; Obogo, Arumosoye, et al., 2021; Arumosoye & Obriki, 2021).

Mutation testing (Jia & Harman, 2011; Papadakis et al., 2019) assesses a test suite by introducing small artificial faults, called mutants, and checking whether the tests detect them. It had long been considered too expensive for industrial use. In 2021, Petrović, Ivanković, Fraser, and Just published two complementary accounts of Google's mutation testing system (Petrović et al., 2021a; Petrović et al., 2021b). The system generated mutants only for changed lines, suppressed mutants unlikely to be useful through heuristics and historical feedback, and surfaced a small number of surviving mutants to developers during code review. The ICSE 2021 study analysed data from millions of mutants and many thousands of developers over several years and found that developers exposed to mutants wrote more tests and that their code was subsequently associated with fewer surviving mutants, suggesting a durable improvement in testing practice (Petrović et al., 2021a). The study also provided evidence that mutants are coupled to real faults, extending earlier controlled evidence on that question (Just, Jalali, Inozemtseva, et al., 2014). The work reframed mutation testing from a batch adequacy metric into an interactive aid integrated in the same review channel as static analysis findings.

Random and search based tools such as Randoop and EvoSuite (Pacheco et al., 2007; Fraser & Arcuri, 2011) had established automated unit test generation for Java, but their tests were often hard for developers to read. Tufano et al. (2020) proposed AthenaTest, which fine-tuned a transformer on pairs of methods and their developer written tests, providing the focal method together with context about its class. Generated tests were judged by developers to be more readable than search based tests while achieving comparable coverage on the evaluated projects. During 2021 this line of research was extended and evaluated further, and Codex style models were observed to produce plausible tests from function signatures and docstrings, foreshadowing the integration of test generation into code assistants (Mbonu et al., 2021b).

Test automation also extended beyond pure software. Comparable principles of repeatable, automated checking appear in non-destructive and non-intrusive inspection regimes for physical equipment (Atta et al., 2021; Atta et al., 2020; Obetta et al., 2021), which, like automated tests, examine a system repeatedly without disrupting its operation. At the organisational level, internal QHSE audit systems (Obogo et al., 2020a) play the role of a final acceptance gate.

Coverage guided greybox fuzzing (Böhme et al., 2016; Manès et al., 2021) remained the most

effective automated technique for finding memory safety and robustness defects in native code, with continuous fuzzing services such as OSS-Fuzz running against hundreds of critical open source projects; an empirical study of OSS-Fuzz bugs published in 2021 characterised how quickly such bugs are found and fixed (Ding & Le Goues, 2021). The AFL++ framework, which consolidated many research improvements into a maintained, extensible fuzzer, became a common base for both research and practice (Fioraldi et al., 2020). A recurrent problem was that fuzzing research was difficult to compare: results depend heavily on benchmarks, trial counts, run durations, and statistical treatment (Klees et al., 2018), and ground-truth benchmarks such as Magma (Hazimeh et al., 2020) and reflections from the community (Böhme et al., 2021) called for more rigorous practice. FuzzBench, presented at ESEC/FSE 2021, addressed this with an open service that runs fuzzers on a common set of real programs, with many trials and long durations, and publishes statistically analysed results on coverage and bug finding (Metzman et al., 2021). FuzzBench is quickly becoming a reference point for fuzzer evaluation, and its experience showed that conclusions drawn from short or few trials were frequently overturned under more rigorous conditions. The security operations counterpart of fuzzing is the continuous exercise of deployed controls; security audit and enterprise risk assessment frameworks (Dosunmu & Ogundele, 2019), intrusion detection and prevention models (Dosunmu & Ogundele, 2020), and risk-based cybersecurity assurance (Fadayomi et al., 2019) bring a similar empirical discipline to security engineering.

A flaky test passes and fails on the same code without any change, typically because of concurrency, timing, ordering dependencies, or reliance on external resources (Luo et al., 2014; Parry et al., 2021). Flaky tests erode trust in continuous integration, waste compute on reruns, and can mask real regressions. Rerunning tests to detect flakiness, as done by tools such as iDFlakies (Lam et al., 2019), is effective but expensive. FlakeFlagger, presented at ICSE 2021, predicted flakiness from static and dynamic features of tests, such as execution time, coverage of recently changed code, use of external resources, and test smells, without repeated reruns (Alshammari et al., 2021). It achieved substantially better prediction than vocabulary based approaches (Pinto et al., 2020) on a large dataset of Java projects, enabling teams to prioritise which tests to rerun or quarantine. The broader message was that flakiness could be managed as a predictable property of tests rather than treated as random noise.

At large scale, running every test for every change is infeasible, particularly in monolithic repositories with billions of lines (Potvin & Levenberg, 2016; Memon et al., 2017), even though continuous integration itself is strongly associated with faster, more reliable delivery (Hilton et al., 2016). Predictive test selection, which uses machine learning on historical results to choose the subset of tests most likely to fail for a given change, had been reported in production at Meta (Machalica et al., 2019), and by 2021 comparable techniques were widely used in large continuous integration systems. The research emphasis shifted toward safety guarantees: quantifying the probability that a skipped test would have caught a regression, and combining learned selection with deterministic dependency analysis so that tests directly touching changed code are never skipped. Platform-specific delivery ecosystems illustrate how such pipelines are assembled in practice, as in systematic reviews of CI/CD strategies, comparative evaluations of DevOps tooling for enterprise CRM platforms, and integration design for platform-centric architectures (Badmus et al., 2018; Badmus et al., 2020; Badmus et al., 2019a).

Site reliability engineering (SRE), as popularised by Google, frames reliability as a measurable product feature managed through service level objectives and error budgets (Beyer et al., 2016). Building on the research programme summarised in Accelerate (Forsgren et al., 2018), the 2021 Accelerate State of DevOps report from DORA extended its established four delivery metrics, deployment frequency, lead time for changes, change failure rate, and time to restore service, with a fifth measure covering reliability, reflecting the degree to which teams meet their own operational targets (Smith et al., 2021). The report found that organisations performing well

on delivery speed also tended to perform well on reliability, supporting the argument that speed and stability are complements rather than trade-offs when supported by automation.

Enterprise practice has developed these ideas in several directions. Frameworks for business continuity and disaster recovery (Fadero et al., 2020), high-throughput engineering of digital payment platforms (Osuji et al., 2021), and performance evaluation of multi-tenant cloud deployments under high concurrency (Odejobi & Ahmed, 2018b) translate availability targets into specific operating models. Machine learning for cloud resource scaling (Ahmed et al., 2020), energy-efficient virtual machine placement (Ahmed & Odejobi, 2018b), constraint satisfaction in cloud resource allocation (Ahmed et al., 2019), and cloud database optimisation (Odejobi et al., 2019) address the resource side of reliability. Maturity models for standardising distributed IT support (Anyanwu, 2019), hybrid support models that balance cost and continuity (Anyanwu, 2020), first-contact resolution in enterprise support (Wikiri, 2021), and standardised change management mechanisms (Uzondu et al., 2021) provide the process layer in which delivery metrics are actually collected and acted upon.

Reliability engineering depends on the ability to observe system behaviour. In February 2021, the OpenTelemetry tracing specification reached version 1.0, providing a vendor neutral standard for distributed traces that is being extended to metrics and logs (OpenTelemetry Authors, 2021). Standardised telemetry allowed quality signals from production, such as error rates on particular code paths, to be linked back to commits and owners, closing the loop between pre-release assurance and post-release monitoring. Related work on enterprise log analytics with automated incident response (Mbonu, Iwuanyanwu, et al., 2019), integrated visualisation for continuous performance monitoring (Ogbole, Okoruwa, Babatope, et al., 2021), and real-time visibility in distribution networks (Ebisi et al., 2021) describes how such signals are aggregated and acted upon in operations.

Chaos engineering tests resilience by deliberately injecting failures into production or production-like environments under controlled conditions (Basiri et al., 2016; Rosenthal & Jones, 2020). In 2021 the practice moved from bespoke tooling at a few large companies to managed cloud services: Amazon Web Services launched its Fault Injection Simulator in March, and Microsoft announced Azure Chaos Studio in preview later in the year. Packaging fault injection as a service lowered the barrier to adoption and encouraged teams to treat resilience experiments as routine tests, run continuously and tied to service level objectives. Business continuity lessons from the COVID-19 pandemic (Fadero et al., 2020) and supply chain resilience frameworks (Anene, 2021) address the same question at the level of whole enterprises.

A significant fraction of severe security defects in large C and C++ codebases are memory safety errors. In 2021, the Android Open Source Project announced support for writing platform components in Rust (Vander Stoep & Hines, 2021), and work began on supporting Rust in the Linux kernel. These decisions represent a different kind of quality strategy: rather than detecting a class of defect after it is written, choose a language or runtime in which the class cannot easily occur. Privacy-centric security engineering for cloud systems (Okoruwa et al., 2020) and identity-centric zero trust (Ogbole, Okoruwa, Fadayomi, et al., 2021) follow the same logic of eliminating classes of weakness through architectural choices. Combined with continuous fuzzing of remaining native code, this approach addressed defects at their source.

The SolarWinds compromise, disclosed in December 2020, showed that an attacker who controls a build system can distribute malicious code through trusted update channels, bypassing source level review entirely. Reviews of open source supply chain attacks (Ohm et al., 2020) and measurements of risk in package ecosystems (Zimmermann et al., 2019; Decan et al., 2018) had already documented how malicious or vulnerable packages propagate. The response in 2021 expanded the definition of codebase reliability to include the provenance of every artefact.

- ***Policy.*** Executive Order 14028, signed on 12 May 2021, directed the development of secure software development standards for federal suppliers and required software bills of materials (SBOMs)

(Executive Office of the President, 2021). In July, the US National Telecommunications and Information Administration published the minimum elements of an SBOM, specifying the data fields, automation support, and practices needed for a useful component inventory (National Telecommunications and Information Administration, 2021).

- ***Build provenance.*** In June, Google introduced SLSA (Supply-chain Levels for Software Artifacts), a graduated framework of requirements for source control, build processes, and provenance metadata designed to make tampering detectable (Lewandowski & Lodato, 2021), building on earlier work on verifiable supply chain metadata such as in-toto (Torres-Arias et al., 2019). Related proposals use blockchain-backed audit trails for cloud configuration and tamper-proof recordkeeping to make changes verifiable (Oshoba et al., 2020; Anichukwueze et al., 2021).
- ***Signing.*** The Sigstore project, launched under the Linux Foundation in 2021, provided free, keyless signing based on short lived certificates tied to identity providers and a public transparency log, removing the key management burden that had discouraged open source maintainers from signing releases (Linux Foundation, 2021).
- ***Dependency vulnerability response.*** In December, CVE-2021-44228 (Log4Shell) disclosed remote code execution in the widely used Log4j logging library (National Institute of Standards and Technology, 2021). Organisations with accurate dependency inventories and automated update tooling identified and patched affected systems quickly; many others spent weeks discovering where the library was used, often as a transitive dependency several levels removed. Integrated cybersecurity and anti-money laundering governance (Fadayomi et al., 2021), cyber risk maturity assessment against sector regulation (Fadero et al., 2021), and data-driven risk evaluation models (Akomolafe & Agu, 2019b) describe how such responses can be organised and prioritised.

Log4Shell made the case for SBOMs and dependency automation more effectively than any policy document. It also exposed a gap in research: most defect detection work evaluates individual programs, whereas the most consequential defect of 2021 propagated across an ecosystem. Reliability engineering at the scale of dependency graphs is emerging as a recognised research problem, closely related to work on IT risk and ISO compliance in digital supply chain transformation, traceability and chain-of-custody integrity, and shared responsibility in global supply chain governance (Mbonu et al., 2020a; Aneke & Adesemoye, 2019; Ominyi et al., 2021).

Many of the ideas reviewed above, such as learned anomaly detection, continuous validation, failure prediction, and automated governance, are being developed in parallel outside mainstream software engineering venues. This section maps that adjacent literature to the software quality agenda. These studies are mostly conceptual frameworks and reviews from applied engineering, security, and management fields, so we treat them as related or semi-related evidence that broadens the context rather than as direct empirical tests of software engineering techniques.

Software reliability engineering borrows much of its vocabulary, including failure modes, mean time to repair, and reliability-centred maintenance, from physical engineering. General treatments of reliability engineering and failure analysis in complex systems (Gideon et al., 2019) and condition-based maintenance (Obetta & Ninduwezuor-Ehiobu, 2018) provide the conceptual foundation, and the shift from reactive to predictive maintenance in mechanical and power systems (Sunday et al., 2020; Adaramola et al., 2018) parallels the move from incident response to defect prediction in software. Inspection and integrity programmes are a close analogue to software test regimes: risk-based inspection and fitness-for-service assessment (Obetta, 2018), integrity management of fired equipment and mooring systems (Obetta, 2019; Obetta & Ninduwezuor-Ehiobu, 2019), and reliability-centred maintenance for static equipment (Obetta, 2020) all aim to find degradation before it becomes failure, while UAV and LiDAR based corridor monitoring (Dagodzo & Ahiaeke Patrick, 2020; Dagodzo & Ahiaeke Patrick, 2021b) plays a

role similar to continuous production monitoring. Incident prevention models for maintenance engineering (Obogo, Obriki, et al., 2021), reviews of safety management systems in maintenance projects (Obogo et al., 2020c), and frameworks for human error causation (Obriki & Arumosoye, 2020) correspond closely to the blameless post-incident review practised in site reliability engineering (Beyer et al., 2016).

Anomaly detection in other domains shares a structure with defect detection: rare positive cases, heavy class imbalance, adversarial adaptation, and a strong need for explanation. Predictive maintenance for e-commerce platforms (Mayo et al., 2021), interpretable early failure prediction for distributed assets (Komi & Adamolekun, 2021), predictive analytics for anti-money laundering (Atakpa, 2021), and VoIP forensic analytics for fraud detection (Mbonu et al., 2021) confront the same imbalance and drift that undermined early vulnerability detectors (Chakraborty et al., 2021). Studies of intrusion detection (Dosunmu & Ogundele, 2020), malware classification in microservices (Idika et al., 2021), and predictive compliance monitoring of systemic safety risks from complaint data (Adeyelu, 2018) make the same methodological points about realistic evaluation that Klees et al. (2018) made for fuzzing.

Security engineering has moved toward continuous, test-like validation that mirrors the quality pipeline. A long-running programme of work traces this progression from insider threat classification (Akeju et al., 2018) and security audit and enterprise risk assessment (Dosunmu & Ogundele, 2019), through intrusion detection and prevention (Dosunmu & Ogundele, 2020) and incident response and digital forensics (Dosunmu & Ogundele, 2021), to risk-based cybersecurity assurance (Fadayomi et al., 2019). In operational technology and critical infrastructure, reviews of NERC CIP compliance for utility OT networks (Adegbite et al., 2020) and zero trust architectures for OT (Ahmed et al., 2021) show how verification and monitoring are being embedded in systems where failures have physical consequences, restating for security the shift from periodic testing to continuous integration (Hilton et al., 2016). At the identity and platform layers, secure cloud messaging architectures (Ahmed & Odejobi, 2018a), offline assessment and implementation of Active Directory (Ladapo et al., 2018; Ladapo et al., 2019), identity and access management for federated systems (Oshoba et al., 2019), and IAM integration across hybrid and multi-cloud environments (Mbonu et al., 2020) provide the configuration-level controls whose misconfiguration is itself a major class of defect.

Quality assurance increasingly operates inside regulatory and governance frameworks. Technology-enabled internal audit (Akomolafe & Agu, 2018), risk-based internal control models (Akomolafe & Agu, 2019a), risk-based business intelligence architecture for financial technology platforms (Mbonu et al., 2021a), and automated risk assessment for regulatory deliverables (Uchechi & Ominyi, 2020) treat compliance as something to be verified continuously by tooling, much as tests verify behaviour. Comparative data protection regulation and secure cloud implementation (Mbonu, Aliliele, et al., 2019) define controls that quality tooling must itself respect. Data governance and interoperability in health systems (Ilodigwe & Adesemoye, 2021), data quality in disease surveillance systems (Ekemezie et al., 2018), and standard operating procedures as change infrastructure (Agboada et al., 2019) underline that data quality is a precondition for every learned quality model.

A final stream concerns the process discipline and decision analytics that surround quality work. Lean Six Sigma adaptations for small enterprises (Ambali et al., 2021), translation of regulatory requirements into compliance workflows (Eyetsemitan et al., 2021), process automation for transparency (Akinleye & Adeyoyin, 2021), and digital transformation with automation in regulated manufacturing (Obukadata et al., 2021a) mirror the move from ad hoc to pipeline-based quality assurance. Risk-based decision-making in good manufacturing practice (Aneke & Adesemoye, 2020), adaptive safety management systems (Adeyelu, 2019), AI-driven risk quantification (Adeyelu, 2020), data-driven occupational risk control (Obriki & Arumosoye, 2018), and AI-based risk stratification for emergency preparedness (Oparah et al., 2021) parallel risk-based

test prioritisation. Data-driven executive decision systems (Lawal & Oduleye, 2019b), machine learning forecasting (Tonoyan et al., 2021a), predictive analytics for demand planning (Aifuwa et al., 2020), and end-to-end visibility and traceability frameworks (Nnabueze et al., 2021) show how the same predictive models that rank risky commits are used to rank operational risk elsewhere.

The synthesis suggests that the advances of 2021 are best understood not as separate tools but as layers of a single assurance pipeline. We therefore propose the Integrated Continuous Quality Assurance (ICQA) model, shown in Figure 1. The model organises the literature into five layers connected by an upward assurance flow and a downward feedback loop.

Layer 1, data and evidence foundation, comprises the representations and data on which every learned quality tool depends: pre-trained code models (Guo et al., 2021; Wang et al., 2021), realistic and deduplicated defect datasets (Zheng et al., 2021; Bhandari et al., 2021), execution-based benchmarks (M. Chen et al., 2021; Hendrycks et al., 2021), and production telemetry (OpenTelemetry Authors, 2021). The vulnerability detection re-evaluation of 2021 (Chakraborty et al., 2021) shows that weaknesses here propagate through every layer above. Data quality and integrity practice from other domains (Ilodigwe & Adesemoye, 2021; Ekemezie et al., 2018) belongs at this layer.

Layer 2, learned and symbolic detection, comprises static analysis (Sadowski et al., 2018; Distefano et al., 2019), graph-based vulnerability detection with statement-level explanation (Li et al., 2021), just-in-time defect prediction (Pornprasit & Tantithamthavorn, 2021; Zeng et al., 2021), and automated review (Tufano et al., 2021). Its outputs are candidate findings ranked by risk, which feed both verification and human review.

Layer 3, verification and repair, comprises the techniques that confirm or correct candidate findings: mutation-based adequacy (Petrović et al., 2021a), fuzzing (Metzman et al., 2021; Manès et al., 2021), flaky-test control (Alshammari et al., 2021; Parry et al., 2021), neural and self-supervised repair (Jiang et al., 2021; Zhu et al., 2021; Yasunaga & Liang, 2021; Allamanis et al., 2021), and patch correctness assessment (Wang et al., 2020; Ye et al., 2021). Execution acts as the arbiter at this layer, which is why compilers and tests can serve as supervisors for learning.

Layer 4, delivery and runtime reliability, comprises continuous integration and test selection (Hilton et al., 2016; Machalica et al., 2019), service level objectives and error budgets (Beyer et al., 2016; Smith et al., 2021), observability, chaos experiments (Basiri et al., 2016), and incident learning. Operational frameworks for continuity, capacity, and support (Fadero et al., 2020; Ahmed et al., 2020; Anyanwu, 2019) sit here.

Layer 5, governance and human oversight, comprises supply chain provenance and policy (Executive Office of the President, 2021; National Telecommunications and Information Administration, 2021; Lewandowski & Lodato, 2021), audit and compliance automation (Akomolafe & Agu, 2018; Uchechi & Ominyi, 2020), and the human factors that determine whether findings are acted upon (Johnson et al., 2013; Onche et al., 2020).

The ***assurance flow*** runs upward: evidence enables detection, detection produces candidates for verification, verified changes are delivered and operated reliably, and governance records and audits the result. The ***feedback loop*** runs downward: escaped defects, incidents, and audit findings become new training data, regression tests, and analyser rules. Break-It-Fix-It (Yasunaga & Liang, 2021), D2A (Zheng et al., 2021), and Google's mutation testing (Petrović et al., 2021a) are each partial instances of this loop, in which the output of one layer improves the data of another. Analogous closed loops in predictive maintenance and inspection (Sunday et al., 2020; Obetta, 2020) and in near-miss reporting (Arumosoye & Obriki, 2019; Obogo, Arumosoye, et al., 2021) show that the pattern generalises beyond software.

The model implies five propositions that can be tested empirically.

- ***P1.*** The measured performance of a Layer 2 detector depends more on the realism of Layer 1 data than on model architecture, as the contrast between curated and realistic vulnerability datasets suggests (Chakraborty et al., 2021; Zeng et al., 2021).

Table 2: ICQA layers, mechanisms, and supporting literature

Layer	*Core mechanisms*	*Representative literature*
L1 Data and evidence	Pre-trained code representations; realistic, deduplicated datasets; execution-based benchmarks; telemetry	(Guo et al., 2021; Bhandari et al., 2021; M. Chen et al., 2021; OpenTelemetry Authors, 2021; Ilodigwe & Adesemoye, 2021)
L2 Detection	Static analysis; graph-based vulnerability detection; just-in-time defect prediction; review automation	(Sadowski et al., 2018; Li et al., 2021; Pornprasit & Tantithamthavorn, 2021; Tufano et al., 2021; Mbonu et al., 2021b)
L3 Verification and repair	Mutation adequacy; fuzzing; flaky-test control; neural and self-supervised repair; patch assessment	(Petrović et al., 2021a; Metzman et al., 2021; Alshammari et al., 2021; Yasunaga & Liang, 2021; Ye et al., 2021)
L4 Delivery and reliability	CI and test selection; SLOs and error budgets; chaos experiments; continuity and capacity	(Hilton et al., 2016; Smith et al., 2021; Basiri et al., 2016; Fadero et al., 2020; Ahmed et al., 2020)
L5 Governance and oversight	Provenance and SBOM; audit and compliance automation; human factors	(Executive Office of the President, 2021; Lewandowski & Lodato, 2021; Akomolafe & Agu, 2018; Uchechi & Ominyi, 2020; Onche et al., 2020)

- ***P2.*** Learned detectors deliver more value when their findings are verified at Layer 3 before reaching developers, because verification removes false positives that erode trust (Johnson et al., 2013; Habib & Pradel, 2018).
- ***P3.*** Organisations that close the feedback loop, turning incidents and escaped defects into tests and training data, improve reliability faster than those that do not (Petrović et al., 2021a; Smith et al., 2021).
- ***P4.*** Findings that are localised and explained at the granularity of a statement or line are acted upon more often than function-level or file-level predictions (Li et al., 2021; Pornprasit & Tantithamthavorn, 2021; Jiarpakdee et al., 2021).
- ***P5.*** Supply chain controls at Layer 5 reduce the time needed to respond to ecosystem-wide vulnerabilities such as Log4Shell (National Institute of Standards and Technology, 2021; Ohm et al., 2020; Torres-Arias et al., 2019).

The literature synthesis and the ICQA model together point to four cross-cutting issues that determine whether the advances of 2021 translate into more dependable software.

The vulnerability detection results of 2021 (Chakraborty et al., 2021) were the clearest example of a general problem: learned quality tools are only as trustworthy as their evaluation data. Duplication between training and test splits, synthetic defects that differ systematically from real ones, and unrealistic class balance can inflate results dramatically. The concern extends to large code models trained on public repositories, which may have seen benchmark problems or their solutions during training. The community responded with deduplication, time based splits, cross-project evaluation, and hand written benchmarks such as HumanEval (M. Chen et al., 2021), and quality-system practices from clinical laboratories offer transferable designs (Yusuf & Abdulrazaq, 2018; Abdulrazaq et al., 2019), but robust contamination control remained an open issue.

Many quality tools are evaluated with proxies: coverage for test quality, test passing for repair correctness, and token overlap for generation. The year saw a clear move toward measures closer to real outcomes, including execution based scoring of generated code (M. Chen et al., 2021), mutation scores coupled to real faults (Petrović et al., 2021a), fuzzing evaluations based on bugs found rather than coverage alone (Metzman et al., 2021), and manual assessment of patch correctness. Each of these is more expensive than the proxy it replaces, which creates tension between rigorous evaluation and the pace of publication.

Generative assistants offered large productivity gains but also a new channel through which defects could enter codebases. The finding that a substantial share of assistant generated code in security relevant scenarios contained known weakness patterns (Pearce et al., 2021) implied that quality pipelines should treat generated code as untrusted input. Static analysis, tests, and review remained necessary, and

there was a strong case for coupling generation with verification, for example by running analysers on suggestions before they are shown. Privacy by design (Ominyi, 2021) adds a legal dimension to this technical problem.

Industrial reports consistently showed that tools succeed when they fit developer workflows. Google's mutation testing (Petrović et al., 2021a), static analysis integrations in code review, and explainable defect predictors (Pornprasit & Tantithamthavorn, 2021) share a design principle: deliver few, relevant, explained findings at the moment a developer is already considering the code. Tools that produce large volumes of unexplained warnings are ignored regardless of accuracy. This principle applies equally to learned models, whose lack of transparency makes explanation (Li et al., 2021) a practical requirement rather than an academic refinement. Research on information-security awareness and records confidentiality (Onche et al., 2020) and on preparing software professionals for international practice (Wikiri, 2019) shows that the same behavioural considerations govern whether security and quality controls are actually followed.

A final observation is that the boundaries between research communities are weakening. Machine learning researchers adopted execution-based evaluation from testing (M. Chen et al., 2021), testing researchers adopted learned prediction from machine learning (Alshammari et al., 2021), and operations practice adopted continuous experimentation from both (Basiri et al., 2016; Smith et al., 2021). The adjacent literature reviewed above shows the same convergence in reliability engineering, security operations, and governance (Gideon et al., 2019; Dosunmu & Ogundele, 2019; Akomolafe & Agu, 2019a), which suggests that methods for handling imbalance, drift, and human triage can be shared more deliberately than they have been.

For researchers, the main implication is methodological. Evaluations of learned quality tools should use deduplicated, time-split, and realistically balanced data, report results on cross-project settings, and prefer outcome measures, such as bugs found, faults detected, or patches accepted, over proxies (Chakraborty et al., 2021; Klees et al., 2018; Tantithamthavorn et al., 2017). The ICQA model also suggests evaluating tools in combination rather than in isolation, for example measuring how detection and repair perform together under a shared verification step.

For practitioners, the evidence supports integrating quality signals into code review rather than issuing batch reports (Sadowski et al., 2018; Petrović et al., 2021a), adopting simple and explainable defect predictors before complex ones (Pornprasit & Tantithamthavorn, 2021; Zeng et al., 2021), quarantining and predicting flaky tests rather than tolerating reruns (Alshammari et al., 2021), treating generated code as untrusted input that must pass the same analysis and tests as human code (Pearce et al., 2021), and maintaining accurate dependency inventories so that ecosystem vulnerabilities can be located quickly (National Telecommunications and Information Administration, 2021; National Institute of Standards and Technology, 2021). Enterprise frameworks for DevOps tooling, cloud resource management, and IT support (Badmus et al., 2020; Ahmed et al., 2020; Anyanwu, 2019) provide practical starting points for organisations building these capabilities.

For policy makers and governance functions, the 2021 measures on secure development, software bills of materials, and build provenance (Executive Office of the President, 2021; National Telecommunications and Information Administration, 2021; Lewandowski & Lodato, 2021) imply that software quality is becoming a matter of regulatory compliance as well as engineering practice. Audit and internal control frameworks (Akomolafe & Agu, 2018; Akomolafe & Agu, 2019a; Uchechi & Ominyi, 2020) will need to incorporate evidence from automated quality tools, and data protection obligations (Mbonu, Aliliele, et al., 2019; Ominyi, 2021) constrain how code and telemetry can be used to train them.

CONCLUSION

The advances of 2021 in automated software quality assurance are best understood as a convergence. Pre-trained models of code supplied a common representation for detection, repair, review, and test generation. Rigorous re-evaluations corrected

optimistic claims and raised standards for evidence. Industrial studies of mutation testing, fuzzing, and flaky tests showed how automated techniques deliver value when integrated into developer workflows. And a series of supply chain incidents, culminating in Log4Shell, expanded the scope of reliability engineering from the individual codebase to the ecosystem on which it depends. Parallel work in reliability engineering, fraud analytics, security operations, and governance shows that these ideas are part of a broader movement toward continuous, evidence-based assurance across engineering disciplines.

The year also left important problems unresolved. Learned detectors still struggle on realistic data, repair systems still produce plausible but incorrect patches, generated code still carries known weaknesses, and the reliability of dependency ecosystems remains difficult to measure. What changed in 2021 was not that these problems were solved, but that the field developed the representations, datasets, evaluation methods, policies, and infrastructure needed to address them in a principled way. Progress in the coming years is likely to build directly on those foundations.

This review has several limitations. First, it is a structured narrative review rather than a full systematic review with formal quality appraisal of every source, so its selection reflects expert judgement and may under-represent work published outside the main venues. Second, the restriction to sources published in or before 2021 means that some 2021 techniques are assessed on early evidence that later work may revise. Third, the consolidated peer literature is drawn largely from applied and conceptual studies in adjacent fields; these are treated as related or semi-related evidence, and their findings are not direct tests of software engineering techniques. Fourth, the ICQA model is conceptual: its layers and propositions are derived from the literature and have not yet been validated empirically. Finally, industrial evidence is concentrated in a small number of large technology organisations, which may limit how far conclusions generalise to smaller teams.

The advances and limitations of 2021, together with the propositions of the ICQA model, suggest the following directions.

- ***Hybrid neuro-symbolic assurance.*** D2A, TFix, and BIFI each paired a symbolic component, such as an analyser, linter, or compiler, with a learned model. Systematic integration of analysers as both data sources and verifiers for large models is a natural continuation.
- ***Execution grounded learning.*** The use of tests and compilers as supervisors suggests that models trained with execution feedback, rather than static text alone, will produce more reliable code and repairs.
- ***Statement level, explainable detection.*** Following IVDetect and JITLine, detectors should localise and explain findings at the granularity developers act upon.
- ***Ecosystem scale reliability.*** Measuring and improving the reliability of dependency graphs, including vulnerability propagation analysis, reachability analysis for transitive dependencies, and automated provenance verification, emerged as a priority after Log4Shell.
- ***Rigorous, contamination aware benchmarks.*** Benchmarks with time based splits, realistic distributions, and execution based scoring are needed for every task in the quality pipeline.
- ***Continuous assurance of assistant generated code.*** Tools that analyse, test, and if necessary repair machine generated code before it reaches a repository will become part of standard development environments.
- ***Cross-domain transfer.*** Predictive maintenance, digital twins, fraud analytics, and continuous control validation (reviewed in Section 3) offer methods for drift handling, imbalance, and human-in-the-loop triage that software quality research has only begun to borrow.
- ***Empirical validation of the ICQA model.*** Field studies and controlled experiments are needed to test propositions P1 to P5, for example by comparing organisations that close the feedback loop with those that do not.

REFERENCES

1. Abdulrazaq, K., Yusuf, H. D., Osinem, H. E., & Eribenne, F. (2019). A model for improving

infection detection accuracy through integrated laboratory information systems. Iconic Research and Engineering Journals, 3(6), 650-675. https://doi.org/10.64388/IREV3I6-1723089

2. Adaramola, T. S., Fadero, S., & Gideon, E. N. (2018). Predictive maintenance and condition monitoring in critical power and energy infrastructure. Iconic Research and Engineering Journals, 2(5), 454-477. https://doi.org/10.64388/IREV2I5-1722489
3. Adegbite, M. P., Adebayo, A., & Ahmed, M. O. (2020). Securing operational technology networks in electric utilities: A systematic review of NERC CIP compliance and architectural threat mitigation. International Journal of Engineering and Modern Technology, 6(3), 103-146. https://doi.org/10.56201/ijemt.vol.6.no3.2020.pg103.146
4. Adeyelu, O. O. (2018). A predictive compliance monitoring framework for detecting systemic safety risks through aviation consumer complaint data. Iconic Research and Engineering Journals, 1(8), 250-276. https://doi.org/10.64388/IREV1I8-1718478
5. Adeyelu, O. O. (2019). An adaptive safety management system implementation model for aerodromes in developing economy contexts. Iconic Research and Engineering Journals, 3(4), 628-654. https://doi.org/10.64388/IREV3I4-1718479
6. Adeyelu, O. O. (2020). An artificial intelligence-driven conceptual model for wildlife strike risk quantification and aircraft airworthiness impact assessment at high-traffic African airports. Iconic Research and Engineering Journals, 4(1), 339-368. https://doi.org/10.64388/IREV4I1-1718482
7. Agboada, P., Ezeako, R., Okoli, I. E., & Uzondu, C. S. (2018). Digital transformation in operations management: A review of advances in automation, analytics, and process redesign. International Journal of Engineering Technology Research & Management, 2(11), 109-134.
8. Agboada, P., Ezeako, R., Okoli, I. E., & Uzondu, C. S. (2019). Standard operating procedures as change infrastructure: A conceptual framework for sustaining operational improvement. Iconic Research and Engineering Journals, 3(6), 706-730. https://doi.org/10.64388/IREV3I6-1723313
9. Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K.-W. (2021). Unified pre-training for program understanding and generation. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 2655-2668).
10. Ahmed, K. S., & Odejobi, O. D. (2018a). Conceptual framework for scalable and secure cloud architectures for enterprise messaging. IRE Journals, 2(1), 1-15.
11. Ahmed, K. S., & Odejobi, O. D. (2018b). Resource allocation model for energy-efficient virtual machine placement in data centers. IRE Journals, 2(3), 1-10.
12. Ahmed, K. S., Odejobi, O. D., & Oshoba, T. O. (2019). Algorithmic model for constraint satisfaction in cloud network resource allocation. IRE Journals, 2(12), 516-532.
13. Ahmed, K. S., Odejobi, O. D., & Oshoba, T. O. (2020). Predictive model for cloud resource scaling using machine learning techniques. Journal of Frontiers in Multidisciplinary Research, 1(1), 173-183.
14. Ahmed, M. O., Adegbite, M. P., & Adebayo, A. (2021). Zero trust architecture for operational technology in North American critical infrastructure: A framework for implementation and resilience optimization. International Journal of Engineering and Modern Technology, 7(1), 66-113. https://doi.org/10.56201/ijemt.vol.7.no1.2021.pg66.113
15. Aifuwa, S. E., Oshoba, T. O., Ogbuefi, E., Ike, P. N., Nnabueze, S. B., & Olatunde-Thorpe, J. (2020). Predictive analytics models enhancing supply chain demand forecasting accuracy and reducing inventory management inefficiencies. International Journal of Multidisciplinary Research and Growth Evaluation, 1(3), 171-181. https://doi.org/10.54660/.IJMRGE.2020.1.3.171-181
16. Akeju, B., Edivri, J., Ogbole, J. I., Okoruwa, P. O., Fadayomi, O., & Abolaji, T. O. (2018). Conceptual model for insider threat classification and risk modeling in complex digital systems. Iconic

Research and Engineering Journals, 1(9), 476-492. https://doi.org/10.64388/IREV1I9-1713778
17. Akinleye, O. K., & Adeyoyin, O. (2021). Process automation framework for enhancing procurement efficiency and transparency. Shodhshauryam, International Scientific Refereed Research Journal, 4(4), 356-387.
18. Akomolafe, O., & Agu, M. U. (2018). A conceptual model for enhancing internal audit quality through technology-enabled risk assessment frameworks. Iconic Research and Engineering Journals, 1(9), 458-475.
19. Akomolafe, O., & Agu, M. U. (2019a). A conceptual framework for developing risk-based internal control models in the insurance and banking sectors. Iconic Research and Engineering Journals, 2(8), 335-354.
20. Akomolafe, O., & Agu, M. U. (2019b). A review of data-driven risk evaluation models for emerging market financial institutions. Iconic Research and Engineering Journals, 3(6), 433-448.
21. Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. ACM Computing Surveys, 51(4), Article 81. https://doi.org/10.1145/3212695
22. Allamanis, M., Jackson-Flux, H., & Brockschmidt, M. (2021). Self-supervised bug detection and repair. In Advances in Neural Information Processing Systems 34 (NeurIPS 2021) (pp. 27865-27876).
23. Alshammari, A., Morris, C., Hilton, M., & Bell, J. (2021). FlakeFlagger: Predicting flakiness without rerunning tests. In Proceedings of the 43rd International Conference on Software Engineering (ICSE 2021) (pp. 1572-1584).
24. Ambali, K. B., Eyetsemitan, R. A., Oyeleye, A. O., & Fadayomi, O. (2021). Lean Six Sigma for small enterprises: A systematic review and Lite-DMAIC adaptation framework for resource-constrained organizations. Iconic Research and Engineering Journals, 5(5), 562-583. https://doi.org/10.64388/IREV5I5-1716957
25. Aneke, O. B., & Adesemoye, A. C. (2019). Toward an integrated conceptual framework for traceability and chain-of-custody integrity in pharmaceutical distribution. Iconic Research and Engineering Journals, 3(4), 655-672.
26. Aneke, O. B., & Adesemoye, A. C. (2020). A conceptual framework for risk-based decision-making in good manufacturing practice environments. Iconic Research and Engineering Journals, 4(6), 325-341.
27. Anene, U. N. (2021). Supply chain resilience and risk management in global distribution networks: A review of advances post-disruption. International Journal of Engineering Technology Research & Management, 5(11), 137-161.
28. Anichukwueze, C. C., Osuji, V. C., & Oguntegbe, E. E. (2021). Blockchain-based architectures for tamper-proof regulatory recordkeeping and real-time audit readiness. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 485-504.
29. Anyanwu, K. C. E. (2019). Developing a maturity model for standardizing IT support operations across geographically distributed multi-office organizations. Iconic Research and Engineering Journals, 3(5)., 580-598.
30. Anyanwu, K. C. E. (2020). Balancing cost and operational continuity in hybrid remote and on-site IT support for multi-branch SMEs. International Journal of Economics and Financial Management, 5(2), 132-148. https://doi.org/10.56201/ijefm.v5.no2.2020.pg132.148
31. Arumosoye, O. M., & Obriki, O. D. (2019). Systematic review of near-miss and hazard observation data utilization in industrial safety management. Iconic Research and Engineering Journals, 3(2), 981-999. https://doi.org/10.64388/IREV3I2-1714417
32. Arumosoye, O. M., & Obriki, O. D. (2021). Organizational learning-based conceptual maturity model for continuous safety performance improvement. International Journal of Multidisciplinary Research and Growth Evaluation, 2(1), 970-980. https://doi.org/10.54660/.IJMRGE.2021.2.1.970-980
33. Asibor, E., & Isong, D. E. (2019). Risk-based process engineering framework for strengthening reliability across complex natural gas infrastructure systems. Iconic Research and Engineering Journals, 3(2), 1020-1060. https://doi.

org/10.64388/IREV3I2-1723153

34. Atakpa, M. I. (2021). A review of predictive analytics models for anti-money laundering detection in commercial banking systems. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(2), 778-804. https://doi.org/10.32628/CSEIT2064813
35. Atta, S., Tofah, P., & Adenuga, O. M. (2021). Advances in non-destructive testing methods for weld integrity evaluation in high-capacity power infrastructure. International Journal of Engineering and Modern Technology, 7(1), 20-47. https://doi.org/10.56201/ijemt.vol.7.no1.2021.pg20.47
36. Atta, S., Tofah, P., Kankam, M. A., & Adenuga, O. M. (2020). A critical review of NDT-based integrity management and corrosion risk assessment strategies for refinery process equipment. International Journal of Engineering and Modern Technology, 6(2), 19-46. https://doi.org/10.56201/ijemt.vol.6.no2.2020.pg19.46
37. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., & Sutton, C. (2021). Program synthesis with large language models. arXiv preprint arXiv:2108.07732. https://doi.org/10.48550/arXiv.2108.07732
38. Bader, J., Scott, A., Pradel, M., & Chandra, S. (2019). Getafix: Learning to fix bugs automatically. Proceedings of the ACM on Programming Languages, 3(OOPSLA), Article 159.
39. Badmus, O., Dosunmu, A. A., & Ozowara, D. E. (2018). A systematic review of CI/CD pipeline strategies in Salesforce DevOps: Tools, practices, and deployment outcomes. Iconic Research and Engineering Journals, 2(6).
40. Badmus, O., Dosunmu, A. A., & Ozowara, D. E. (2019a). A conceptual model for ETL design and data integration in Salesforce-centric enterprise architectures. Iconic Research and Engineering Journals, 3(5).
41. Badmus, O., Dosunmu, A. A., & Ozowara, D. E. (2019b). A governance framework for Salesforce platform management in regulated healthcare environments. Iconic Research and Engineering Journals, 3(6).
42. Badmus, O., Dosunmu, A. A., Ozowara, D. E., & Anunagba, C. O. (2020). A comparative framework for Salesforce DevOps tooling: Evaluating Copado, Flosum, and Salesforce DX across enterprise deployment contexts. International Journal of Multidisciplinary Research and Growth Evaluation, 1(5), 1021-1031. https://doi.org/10.54660/.IJMRGE.2020.1.5.1021-1031
43. Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., & Rosenthal, C. (2016). Chaos engineering. IEEE Software, 33(3), 35-41.
44. Berabi, B., He, J., Raychev, V., & Vechev, M. (2021). TFix: Learning to fix coding errors with a text-to-text transformer. In Proceedings of the 38th International Conference on Machine Learning (ICML 2021) (pp. 780-791). PMLR.
45. Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., & Engler, D. (2010). A few billion lines of code later: Using static analysis to find bugs in the real world. Communications of the ACM, 53(2), 66-75.
46. Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (Eds.). (2016). Site reliability engineering: How Google runs production systems. O'Reilly Media.
47. Bhandari, G., Naseer, A., & Moonen, L. (2021). CVEfixes: Automated collection of vulnerabilities and their fixes from open-source software. In Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2021) (pp. 30-39). https://doi.org/10.1145/3475960.3475985
48. Böhme, M., Cadar, C., & Roychoudhury, A. (2021). Fuzzing: Challenges and reflections. IEEE Software, 38(3), 79-86.
49. Böhme, M., Pham, V.-T., & Roychoudhury, A. (2016). Coverage-based greybox fuzzing as Markov chain. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16) (pp. 1032-1043).
50. Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P.,

Papakonstantinou, I., Purbrick, J., & Rodriguez, D. (2015). Moving fast with software verification. In NASA Formal Methods (NFM 2015), Lecture Notes in Computer Science, Vol. 9058 (pp. 3-11). Springer.

51. Chakraborty, S., Krishna, R., Ding, Y., & Ray, B. (2021). Deep learning based vulnerability detection: Are we there yet? IEEE Transactions on Software Engineering. Advance online publication. https://doi.org/10.1109/TSE.2021.3087402
52. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., ... Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. https://doi.org/10.48550/arXiv.2107.03374
53. Chen, Z., Kommrusch, S., Tufano, M., Pouchet, L.-N., Poshyvanyk, D., & Monperrus, M. (2021). SequenceR: Sequence-to-sequence learning for end-to-end program repair. IEEE Transactions on Software Engineering, 47(9), 1943-1959.
54. Dagodzo, D., & Ahiaeke Patrick, M. C. (2020). UAV-based pipeline and corridor monitoring: A review of current practices and emerging technologies. Iconic Research and Engineering Journals, 3(10), 574-597. https://doi.org/10.64388/IREV3I10-1716084
55. Dagodzo, D., & Ahiaeke Patrick, M. C. (2021b). An integrated framework for UAV, LiDAR, and GIS in infrastructure corridor management. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 497-524. https://doi.org/10.32628/CSEIT217566
56. Decan, A., Mens, T., & Constantinou, E. (2018). On the impact of security vulnerabilities in the npm package dependency network. In Proceedings of the 15th International Conference on Mining Software Repositories (MSR 2018) (pp. 181-191).
57. Dinella, E., Dai, H., Li, Z., Naik, M., Song, L., & Wang, K. (2020). Hoppity: Learning graph transformations to detect and fix bugs in programs. In Proceedings of the International Conference on Learning Representations (ICLR 2020).
58. Ding, Z. Y., & Le Goues, C. (2021). An empirical study of OSS-Fuzz bugs. In Proceedings of the 18th International Conference on Mining Software Repositories (MSR 2021) (pp. 131-142).
59. Distefano, D., Fähndrich, M., Logozzo, F., & O'Hearn, P. W. (2019). Scaling static analyses at Facebook. Communications of the ACM, 62(8), 62-70.
60. Dosunmu, A. A., & Ogundele, P. O. (2019). Security audit and enterprise risk assessment frameworks for resilient information systems. Iconic Research and Engineering Journals, 3(5), 434-447. https://doi.org/10.64388/IREV3I5-1713225
61. Dosunmu, A. A., & Ogundele, P. O. (2020). Intrusion detection and prevention models for enhancing organizational cyber defense effectiveness. Iconic Research and Engineering Journals, 4(6), 310-324. https://doi.org/10.64388/IREV4I6-1713226
62. Dosunmu, A. A., & Ogundele, P. O. (2021). Incident response and digital forensics strategies for rapid cyber attack containment. Gyanshauryam, International Scientific Refereed Research Journal, 4(4), 239-258.
63. Drain, D., Clement, C. B., Serrato, G., & Sundaresan, N. (2021). DeepDebug: Fixing Python bugs using stack traces, backtranslation, and code skeletons. arXiv preprint arXiv:2105.09352.
64. Ebisi, C. A., Sam-Bulya, N. J., Eyo-Udo, N. L., & Anene, U. (2021). Real-time order visibility for lead-time compression in global heavy-equipment distribution networks. International Journal of Multidisciplinary Evolutionary Research, 2(1), 106-123. https://doi.org/10.54660/IJMER.2021.2.1.106-123
65. Ekemezie, U. C., Ohunyon, G., & Sydney, E. (2018). Evaluating IDSR implementation in Benue State, Nigeria: DHIS2 data quality and surveillance capacity. Iconic Research and Engineering Journals, 2(6).
66. Executive Office of the President. (2021). Executive Order 14028: Improving the nation's cybersecurity. Federal Register, 86(93), 26633-26647.
67. Eyetsemitan, R. A., Ambali, K. B., Oyeleye, A. O., & Fadayomi, O. (2021). Translating tax and regulatory requirements into SME

compliance workflows: A conceptual framework for implementing IAS 12, VAT, PAYE, and withholding tax. Iconic Research and Engineering Journals, 4(11), 621-641. https://doi.org/10.64388/IREV4I11-1716956

68. Fadayomi, O., Abolaji, T. O., Edivri, J., Ogbole, J. I., Okoruwa, P. O., & Akeju, B. (2019). Risk-based cybersecurity assurance and data availability: Limitations, advances and future research opportunities. Iconic Research and Engineering Journals, 2(12), 602-617. https://doi.org/10.64388/IREV2I12-1713779
69. Fadayomi, O., Bello, A. D., Elebe, O., Hammed, N. I., & Omoegun, G. O. (2021). An integrated cybersecurity and anti-money laundering governance framework for financial crime prevention [Preprint]. ResearchGate.
70. Fadero, S., Jooda, D., Oyesiji, S. O., & Nwakamma, S. (2020). Business continuity and disaster recovery preparedness in African financial services: Lessons from the COVID-19 pandemic. Iconic Research and Engineering Journals, 3(9), 525-544. https://doi.org/10.64388/IREV3I9-1722620
71. Fadero, S., Nwakamma, S., Jooda, D., & Oyesiji, S. O. (2021). Cyber risk management maturity in Nigeria's insurance sector: An assessment against the NAICOM and NDPR regulatory landscape. Iconic Research and Engineering Journals, 5(2), 401-420. https://doi.org/10.64388/IREV5I2-1722621
72. Fan, J., Li, Y., Wang, S., & Nguyen, T. N. (2020). A C/C++ code vulnerability dataset with code changes and CVE summaries. In Proceedings of the 17th International Conference on Mining Software Repositories (MSR 2020) (pp. 508-512).
73. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. In Findings of the Association for Computational Linguistics: EMNLP 2020 (pp. 1536-1547).
74. Fioraldi, A., Maier, D., Eißfeldt, H., & Heuse, M. (2020). AFL++: Combining incremental steps of fuzzing research. In Proceedings of the 14th USENIX Workshop on Offensive Technologies (WOOT 20).
75. Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The science of lean software and DevOps. IT Revolution Press.
76. Fraser, G., & Arcuri, A. (2011). EvoSuite: Automatic test suite generation for object-oriented software. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011) (pp. 416-419).
77. Gideon, E. N., Adaramola, T. S., & Fadero, S. (2019). Reliability engineering and failure analysis in complex engineering systems. Iconic Research and Engineering Journals, 2(11), 703-727. https://doi.org/10.64388/IREV2I11-1722490
78. Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., ... Zhou, M. (2021). GraphCodeBERT: Pre-training code representations with data flow. In Proceedings of the International Conference on Learning Representations (ICLR 2021).
79. Habib, A., & Pradel, M. (2018). How many of all bugs do we find? A study of static bug detectors. In Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2018) (pp. 317-328).
80. Hazimeh, A., Herrera, A., & Payer, M. (2020). Magma: A ground-truth fuzzing benchmark. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 4(3), Article 49.
81. Hellendoorn, V. J., & Devanbu, P. (2017). Are deep neural networks the best choice for modeling source code? In Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017) (pp. 763-773).
82. Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., & Steinhardt, J. (2021). Measuring coding challenge competence with APPS. In Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks.
83. Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. (2016). Usage, costs, and benefits of continuous integration in open-source projects. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016) (pp. 426-437).

84. Hindle, A., Barr, E. T., Su, Z., Gabel, M., & Devanbu, P. (2012). On the naturalness of software. In Proceedings of the 34th International Conference on Software Engineering (ICSE 2012) (pp. 837-847).
85. Hoang, T., Dam, H. K., Kamei, Y., Lo, D., & Ubayashi, N. (2019). DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction. In Proceedings of the 16th International Conference on Mining Software Repositories (MSR 2019) (pp. 34-45).
86. Hoang, T., Kang, H. J., Lo, D., & Lawall, J. (2020). CC2Vec: Distributed representations of code changes. In Proceedings of the 42nd International Conference on Software Engineering (ICSE 2020) (pp. 518-529).
87. Idika, C. N., Salami, E. O., Ijiga, O. M., & Enyejo, L. A. (2021). Deep learning driven malware classification for cloud-native microservices in edge computing architectures. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(4).
88. Ilodigwe, L., & Adesemoye, A. C. (2021). The data backbone of health system transformation: A governance and architecture framework for interoperability, data quality, and advanced analytics at national scale. International Journal of Health and Pharmaceutical Research, 6(2), 52-76. https://doi.org/10.56201/ijhpr.vol.6.no2.2021.pg52.76
89. Jia, Y., & Harman, M. (2011). An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering, 37(5), 649-678.
90. Jiang, N., Lutellier, T., & Tan, L. (2021). CURE: Code-aware neural machine translation for automatic program repair. In Proceedings of the 43rd International Conference on Software Engineering (ICSE 2021) (pp. 1161-1173).
91. Jiarpakdee, J., Tantithamthavorn, C. K., & Grundy, J. (2021). Practitioners' perceptions of the goals and visual explanations of defect prediction models. In Proceedings of the 18th International Conference on Mining Software Repositories (MSR 2021) (pp. 432-443).
92. Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? In Proceedings of the 35th International Conference on Software Engineering (ICSE 2013) (pp. 672-681).
93. Just, R., Jalali, D., & Ernst, M. D. (2014). 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 2014) (pp. 437-440).
94. Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R., & Fraser, G. (2014). Are mutants a valid substitute for real faults in software testing? In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014) (pp. 654-665).
95. Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2013). A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering, 39(6), 757-773.
96. Kanade, A., Maniatis, P., Balakrishnan, G., & Shi, K. (2020). Learning and evaluating contextual embedding of source code. In Proceedings of the 37th International Conference on Machine Learning (ICML 2020) (pp. 5110-5121). PMLR.
97. Klees, G., Ruef, A., Cooper, B., Wei, S., & Hicks, M. (2018). Evaluating fuzz testing. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18) (pp. 2123-2138).
98. Komi, N. M., & Adamolekun, A. (2021). Interpretable machine learning for early failure prediction in distributed renewable energy assets. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 1015-1038. https://doi.org/10.54660/.IJMRGE.2021.2.6.1015-1038
99. Ladapo, O. O., Dosunmu, A. A., Jooda, D., & Abolaji, T. O. (2018). Lessons learned from offline assessment of security-critical systems: The case of Microsoft Active Directory. Iconic Research and Engineering Journals, 2(6), 277-299. https://doi.org/10.64388/IREV2I6-1717205

100. Ladapo, O. O., Jooda, D., Dosunmu, A. A., & Abolaji, T. O. (2019). Implementation of Active Directory for efficient management of enterprise networks. Iconic Research and Engineering Journals, 3(4), 608-627. https://doi.org/10.64388/IREV3I4-1717206
101. Lam, W., Oei, R., Shi, A., Marinov, D., & Xie, T. (2019). iDFlakies: A framework for detecting and partially classifying flaky tests. In Proceedings of the 12th IEEE Conference on Software Testing, Validation and Verification (ICST 2019) (pp. 312-322).
102. Lawal, O. A., & Oduleye, T. E. (2019b). Conceptualizing data driven executive decision systems for strategic financial planning. Iconic Research and Engineering Journals, 3(3).
103. Le Goues, C., Pradel, M., & Roychoudhury, A. (2019). Automated program repair. Communications of the ACM, 62(12), 56-65.
104. Lewandowski, K., & Lodato, M. (2021, June 16). Introducing SLSA, an end-to-end framework for supply chain integrity. Google Security Blog.
105. Li, Y., Wang, S., & Nguyen, T. N. (2021). Vulnerability detection with fine-grained interpretations. In Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021) (pp. 292-303). https://doi.org/10.1145/3468264.3468597
106. Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2018b). SySeVR: A framework for using deep learning to detect software vulnerabilities. arXiv preprint arXiv:1807.06756.
107. Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., & Zhong, Y. (2018a). VulDeePecker: A deep learning-based system for vulnerability detection. In Proceedings of the Network and Distributed System Security Symposium (NDSS 2018).
108. Linux Foundation. (2021, March 9). Linux Foundation announces free sigstore signing service to confirm origin and authenticity of software [Press release].
109. Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., ... Liu, S. (2021). CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks.
110. Luo, Q., Hariri, F., Eloussi, L., & Marinov, D. (2014). An empirical analysis of flaky tests. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014) (pp. 643-653).
111. Lutellier, T., Pham, H. V., Pang, L., Li, Y., Wei, M., & Tan, L. (2020). CoCoNuT: Combining context-aware neural translation models using ensemble for program repair. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2020) (pp. 101-114).
112. Machalica, M., Samylkin, A., Porth, M., & Chandra, S. (2019). Predictive test selection. In Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2019) (pp. 91-100).
113. Manès, V. J. M., Han, H., Han, C., Cha, S. K., Egele, M., Schwartz, E. J., & Woo, M. (2021). The art, science, and engineering of fuzzing: A survey. IEEE Transactions on Software Engineering, 47(11), 2312-2331.
114. Marginean, A., Bader, J., Chandra, S., Harman, M., Jia, Y., Mao, K., Mols, A., & Scott, A. (2019). SapFix: Automated end-to-end repair at scale. In Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2019) (pp. 269-278).
115. Mayo, W., Ogbole, J. I., Okoruwa, P. O., & Babatope, O. M. (2021). Designing an AI-predictive maintenance model for e-commerce systems using machine learning and cloud analytics. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 416-440. https://doi.org/10.32628/IJSRCSEIT
116. Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Oluoha, O. M. (2018). A conceptual framework for legal and ethical risk modeling in enterprise data protection governance systems. Iconic Research and Engineering Journals, 2(2), 207-226. https://doi.org/10.64388/IREV2I2-1714911
117. Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., &

Uzoka, E. (2020). A review of identity and access management integration strategies in hybrid and multi cloud environments. International Journal of Multidisciplinary Research and Growth Evaluation, 1(5), 795-810. https://doi.org/10.54660/.IJMRGE.2020.1.5.795-810

118. Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Uzoka, E. (2021a). A conceptual framework for risk based business intelligence architecture in financial technology platforms. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 731-746. https://doi.org/10.54660/.IJMRGE.2021.2.6.731-746
119. Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Uzoka, E. (2021b). Advances in artificial intelligence techniques for secure software testing and automated regression control mechanisms. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 468-496. https://doi.org/10.32628/CSEIT217565
120. Mbonu, I. S., Aliliele, C., Uzoka, E., & Oluoha, O. M. (2019). A review of comparative data protection regulations and secure cloud implementation strategies across jurisdictions. Iconic Research and Engineering Journals, 2(9), 482-501. https://doi.org/10.64388/IREV2I9-1714912
121. Mbonu, I. S., Iwuanyanwu, U., Aliliele, C., & Uzoka, E. (2020a). A conceptual framework for agile supply chain digital transformation with embedded IT risk and ISO compliance controls. Iconic Research and Engineering Journals, 3(11), 566-593. https://doi.org/10.64388/IREV3I11-1714916
122. Mbonu, I. S., Iwuanyanwu, U., Aliliele, C., & Uzoka, E. (2020b). Advances in infrastructure as code governance for secure terraform based enterprise cloud deployments. International Journal of Multidisciplinary Research and Growth Evaluation, 1(5), 811-828. https://doi.org/10.54660/.IJMRGE.2020.1.5.811-828
123. Mbonu, I. S., Iwuanyanwu, U., Aliliele, C., & Uzoka, E. (2021). A review of VoIP forensic analytics models for financial fraud detection and regulatory compliance monitoring. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 711-730. https://doi.org/10.54660/.IJMRGE.2021.2.6.711-730
124. Mbonu, I. S., Iwuanyanwu, U., Uzoka, E., & Oluoha, O. M. (2019). Advances in enterprise log analytics and automated incident response architectures using Python and SIEM platforms. Iconic Research and Engineering Journals, 3(2), 1000-1019. https://doi.org/10.64388/IREV3I2-1714915
125. Memon, A., Gao, Z., Nguyen, B., Dhanda, S., Nickell, E., Siemborski, R., & Micco, J. (2017). Taming Google-scale continuous testing. In Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2017) (pp. 233-242).
126. Metzman, J., Szekeres, L., Simon, L., Sprabery, R., & Arya, A. (2021). FuzzBench: An open fuzzer benchmarking platform and service. In Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021) (pp. 1393-1403).
127. Monperrus, M. (2018). Automatic software repair: A bibliography. ACM Computing Surveys, 51(1), Article 17.
128. National Institute of Standards and Technology. (2021). CVE-2021-44228 detail. National Vulnerability Database.
129. National Telecommunications and Information Administration. (2021). The minimum elements for a software bill of materials (SBOM). U.S. Department of Commerce.
130. Nnabueze, S. B., Ike, P. N., Olatunde-Thorpe, J., Aifuwa, S. E., Oshoba, T. O., Ogbuefi, E., & Akokodaripon, D. (2021). End-to-end visibility frameworks improving transparency, compliance, and traceability across complex global supply chain operations. International Journal of Multidisciplinary Futuristic Development, 2(2), 50-60. https://doi.org/10.54660/IJMFD.2021.2.2.50-60
131. Obetta, P. C. (2018). A risk-based inspection concept for fitness-for-service assessment of ageing pressure vessels on offshore joint venture facilities. Iconic Research and Engineering Journals, 2(5), 497-520. https://doi.org/10.64388/

IREV2I5-1722578

132. Obetta, P. C. (2019). Reliability and integrity management of fired heaters, boilers and furnaces in upstream oil and gas: A review. Iconic Research and Engineering Journals, 3(5), 599-626. https://doi.org/10.64388/IREV3I5-1722580
133. Obetta, P. C. (2020). A reliability-centered maintenance (RCM) concept for static mechanical equipment in offshore oil and gas operations. International Journal of Multidisciplinary Futuristic Development, 1(1), 74-91. https://doi.org/10.54660/IJMFD.2020.1.1.74-91
134. Obetta, P. C., & Ninduwezuor-Ehiobu, N. (2018). A condition-based maintenance concept for CALM buoy bearings and swivels to improve offshore crude export availability. Iconic Research and Engineering Journals, 2(6), 385-410. https://doi.org/10.64388/IREV2I6-1722579
135. Obetta, P. C., & Ninduwezuor-Ehiobu, N. (2019). Integrity assurance of FSO turret mooring systems: A review of inspection, monitoring and life-extension strategies. Iconic Research and Engineering Journals, 3(6), 599-629. https://doi.org/10.64388/IREV3I6-1722581
136. Obetta, P. C., Ninduwezuor-Ehiobu, N., & Olisakwe, H. C. (2021). Advances in non-intrusive inspection of pressure vessels: Reducing shutdown frequency and OPEX in oil and gas operations. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 1137-1153. https://doi.org/10.54660/.IJMRGE.2021.2.6.1137-1153
137. Obogo, S. F., Arumosoye, O. M., & Obriki, O. D. (2020a). Advances in internal QHSE audit systems for industrial engineering operations. Iconic Research and Engineering Journals, 4(4), 399-417. https://doi.org/10.64388/IREV4I4-1715499
138. Obogo, S. F., Arumosoye, O. M., & Obriki, O. D. (2020c). Critical review of occupational safety management systems in oil and gas maintenance projects. Shodhshauryam, International Scientific Refereed Research Journal, 3(4), 145-166. https://doi.org/10.32628/SHISRRJ214442
139. Obogo, S. F., Arumosoye, O. M., & Obriki, O. D. (2021). Advances in proactive hazard recognition and near miss reporting systems. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 835-846. https://doi.org/10.54660/IJMRGE.2021.2.6.835-846
140. Obogo, S. F., Obriki, O. D., & Arumosoye, O. M. (2021). Conceptual model for incident prevention in industrial maintenance engineering environments. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 847-858. https://doi.org/10.54660/IJMRGE.2021.2.6.847-858
141. Obriki, O. D., & Arumosoye, O. M. (2018). Conceptual modeling of data-driven occupational safety risk control in large-scale energy infrastructure projects. Iconic Research and Engineering Journals, 1(7), 169-189. https://doi.org/10.64388/IREV1I7-1714414
142. Obriki, O. D., & Arumosoye, O. M. (2020). Conceptual framework for human error causation in high-risk construction and industrial activities. International Journal of Multidisciplinary Research and Growth Evaluation, 1(5), 715-727. https://doi.org/10.54660/.IJMRGE.2020.1.5.715-727
143. Obukadata, V. O., Awulu, I., & Williams, I. (2021a). Digital transformation, data analytics, and automation driving operational performance in pharmaceutical manufacturing. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 1173-1194. https://doi.org/10.54660/.IJMRGE.2021.2.6.1173-1194
144. Odejobi, O. D., & Ahmed, K. S. (2018b). Performance evaluation model for multi-tenant Microsoft 365 deployments under high concurrency. IRE Journals, 1(11), 92-107.
145. Odejobi, O. D., Hammed, N. I., & Ahmed, K. S. (2019). Approximation complexity model for cloud-based database optimization problems. IRE Journals, 2(9), 1-10.
146. Ogbole, J. I., Okoruwa, P. O., Babatope, O. M., & Oyewole, T. (2021). Developing an integrated data visualization model for continuous business performance monitoring and optimization. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 441-467. https://doi.

org/10.32628/IJSRCSEIT

147. Ogbole, J. I., Okoruwa, P. O., Fadayomi, O., Abolaji, T. O., Edivri, J., & Akeju, B. (2021). Conceptual model for identity-centric zero trust architecture in enterprise security governance. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 393-415. https://doi.org/10.32628/IJSRCSEIT217562
148. Ohm, M., Plate, H., Sykosch, A., & Meier, M. (2020). Backstabber's knife collection: A review of open source software supply chain attacks. In Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2020), Lecture Notes in Computer Science, Vol. 12223 (pp. 23-43). Springer.
149. Okoruwa, P. O., Fadayomi, O., Akeju, B., Edivri, J., Ogbole, J. I., & Abolaji, T. O. (2020). Conceptual model for privacy-centric security engineering in digital and cloud computing systems. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 371-389.
150. Ominyi, M. (2021). Toward a conceptual framework for embedding privacy by design in corporate restructuring and transaction structuring. Journal of Accounting and Financial Management, 7(5), 152-173. https://doi.org/10.56201/jafm.vol.7.no5.2021.p152.173
151. Ominyi, M., Uchechi, N., Anichukwueze, C. C., & Jones, B. C. (2021). Shared responsibility and multi-jurisdictional alignment in global supply chain governance. International Journal of Multidisciplinary Evolutionary Research, 2(2), 131-147. https://doi.org/10.54660/IJMER.2021.2.2.131-147
152. Onche, V. O., Ogbonna, C. S., & Adegbite, M. P. (2020). Information-security awareness and records confidentiality in Nigerian university registries: A behavioural assessment. International Journal of Multidisciplinary Research and Growth Evaluation, 1(5), 1032-1044. https://doi.org/10.54660/.IJMRGE.2020.1.5.1032-1044
153. Oparah, O. S., Ezeh, F. E., Olatunji, G. I., & Ajayi, O. O. (2021). AI-based risk stratification framework for large-scale public health emergency preparedness and response planning. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(1), 332-366.
154. OpenTelemetry Authors. (2021). OpenTelemetry specification: Tracing 1.0. Cloud Native Computing Foundation.
155. Oshoba, T. O., Hammed, N. I., & Odejobi, O. D. (2019). Secure identity and access management model for distributed and federated systems. IRE Journals, 3(4), 550-567.
156. Oshoba, T. O., Hammed, N. I., & Odejobi, O. D. (2020). Blockchain-enabled compliance and audit trail model for cloud configuration management. Journal of Frontiers in Multidisciplinary Research, 1(1), 193-201.
157. Osuji, V. C., Okafor, C. M., & Dako, O. F. (2021). Engineering high-throughput digital collections platforms for multibillion-dollar payment ecosystems. Shodhshauryam, International Scientific Refereed Research Journal, 4(4), 315-335.
158. Pacheco, C., Lahiri, S. K., Ernst, M. D., & Ball, T. (2007). Feedback-directed random test generation. In Proceedings of the 29th International Conference on Software Engineering (ICSE 2007) (pp. 75-84).
159. Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Le Traon, Y., & Harman, M. (2019). Mutation testing advances: An analysis and survey. Advances in Computers, 112, 275-378.
160. Parry, O., Kapfhammer, G. M., Hilton, M., & McMinn, P. (2021). A survey of flaky tests. ACM Transactions on Software Engineering and Methodology, 31(1), Article 17.
161. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2021). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. arXiv preprint arXiv:2108.09293.
162. Petrović, G., Ivanković, M., Fraser, G., & Just, R. (2021a). Does mutation testing improve testing practices? In Proceedings of the 43rd International Conference on Software Engineering (ICSE 2021) (pp. 910-921).
163. Petrović, G., Ivanković, M., Fraser, G., & Just, R. (2021b). Practical mutation testing at scale: A view

from Google. IEEE Transactions on Software Engineering. Advance online publication. https://doi.org/10.1109/TSE.2021.3107634

164. Pinto, G., Miranda, B., Dissanayake, S., d'Amorim, M., Treude, C., & Bertolino, A. (2020). What is the vocabulary of flaky tests? In Proceedings of the 17th International Conference on Mining Software Repositories (MSR 2020) (pp. 492-502).
165. Pornprasit, C., & Tantithamthavorn, C. (2021). JITLine: A simpler, better, faster, finer-grained just-in-time defect prediction. In Proceedings of the 18th International Conference on Mining Software Repositories (MSR 2021) (pp. 369-379).
166. Potvin, R., & Levenberg, J. (2016). Why Google stores billions of lines of code in a single repository. Communications of the ACM, 59(7), 78-87.
167. Puri, R., Kung, D. S., Janssen, G., Zhang, W., Domeniconi, G., Zolotov, V., ... Reiss, F. (2021). CodeNet: A large-scale AI for code dataset for learning a diversity of coding tasks. In Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks.
168. Rosenthal, C., & Jones, N. (2020). Chaos engineering: System resiliency in practice. O'Reilly Media.
169. Russell, R., Kim, L., Hamilton, L., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P., & McConley, M. (2018). Automated vulnerability detection in source code using deep representation learning. In Proceedings of the 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 757-762).
170. Sadowski, C., Aftandilian, E., Eagle, A., Miller-Cushon, L., & Jaspan, C. (2018). Lessons from building static analysis tools at Google. Communications of the ACM, 61(4), 58-66.
171. Smith, D., Villalba, D., Irvine, M., Stanke, D., & Harvey, N. (2021). Accelerate State of DevOps 2021. DORA and Google Cloud.
172. Sunday, E. A., Omoegun, G. O., Essien, M. A., & Oluokun, O. A. (2020). Transitioning from reactive to predictive maintenance in mechanical systems. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 6(6), 425-447.
173. Svyatkovskiy, A., Deng, S. K., Fu, S., & Sundaresan, N. (2020). IntelliCode Compose: Code generation using transformer. In Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020) (pp. 1433-1443).
174. Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2017). An empirical comparison of model validation techniques for defect prediction models. IEEE Transactions on Software Engineering, 43(1), 1-18.
175. Tian, H., Liu, K., Kaboré, A. K., Koyuncu, A., Li, L., Klein, J., & Bissyandé, T. F. (2020). Evaluating representation learning of code changes for predicting patch correctness in program repair. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE 2020) (pp. 981-992).
176. Tonoyan, A., Dada, O., & Ayivi-Donkor, S. S. (2021a). Advances in demand forecasting: Machine learning algorithms for revenue projection and financial planning accuracy. Gyanshauryam, International Scientific Refereed Research Journal, 4(1), 282-317.
177. Torres-Arias, S., Afzali, H., Kuppusamy, T. K., Curtmola, R., & Cappos, J. (2019). in-toto: Providing farm-to-table guarantees for bits and bytes. In Proceedings of the 28th USENIX Security Symposium (pp. 1393-1410).
178. Tufano, M., Drain, D., Svyatkovskiy, A., Deng, S. K., & Sundaresan, N. (2020). Unit test case generation with transformers and focal context. arXiv preprint arXiv:2009.05617.
179. Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D., & Bavota, G. (2021). Towards automating code review activities. In Proceedings of the 43rd International Conference on Software Engineering (ICSE 2021) (pp. 163-174).
180. Uchechi, N., & Ominyi, M. (2020). Advances in automated risk assessment frameworks for regulatory deliverables in upstream energy operations. International Journal of Multidisciplinary Evolutionary Research, 1(2), 158-175. https://doi.org/10.54660/IJMER.2020.1.2.158-175

181. Uchechi, N., Ominyi, M., Anichukwueze, C. C., & Jones, B. C. (2021). Standardizing contract review workflows in enterprise legal operations: A systematic review. International Journal of Multidisciplinary Evolutionary Research, 2(2), 148-166. https://doi.org/10.54660/IJMER.2021.2.2.148-166
182. Uzondu, C. S., Ibeh, A. V., Alugwe, S., & Awulu, I. (2021). Standardizing change management mechanisms in large organizations: A review of advances in adoption and sustainment. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 1226-1243. https://doi.org/10.54660/.IJMRGE.2021.2.6.1226-1243
183. Vander Stoep, J., & Hines, S. (2021, April 6). Rust in the Android platform. Google Security Blog.
184. Wang, S., Liu, T., & Tan, L. (2016). Automatically learning semantic features for defect prediction. In Proceedings of the 38th International Conference on Software Engineering (ICSE 2016) (pp. 297-308).
185. Wang, S., Wen, M., Lin, B., Wu, H., Qin, Y., Zou, D., Mao, X., & Jin, H. (2020). Automated patch correctness assessment: How far are we? In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE 2020) (pp. 968-980).
186. Wang, Y., Wang, W., Joty, S., & Hoi, S. C. H. (2021). CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (pp. 8696-8708). Association for Computational Linguistics.
187. Wattanakriengkrai, S., Thongtanunam, P., Tantithamthavorn, C., Hata, H., & Matsumoto, K. (2020). Predicting defective lines using a model-agnostic technique. IEEE Transactions on Software Engineering. Advance online publication.
188. Wikiri, M. (2019). Global technology talent development in emerging markets: A conceptual framework for preparing software professionals for international employment. Iconic Research and Engineering Journals, 3(5), 651-671. https://doi.org/10.64388/IREV3I5-1723088
189. Wikiri, M. (2021). Improving first-contact resolution in enterprise technology support: A workforce competency and service-quality framework. International Journal of Multidisciplinary Research and Growth Evaluation, 2(6), 1212-1225. https://doi.org/10.54660/.IJMRGE.2021.2.6.1212-1225
190. Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. (2014). Modeling and discovering vulnerabilities with code property graphs. In Proceedings of the 2014 IEEE Symposium on Security and Privacy (pp. 590-604).
191. Yasunaga, M., & Liang, P. (2021). Break-It-Fix-It: Unsupervised learning for program repair. In Proceedings of the 38th International Conference on Machine Learning (ICML 2021) (pp. 11941-11952). PMLR.
192. Ye, H., Martinez, M., & Monperrus, M. (2021). Automated patch assessment for program repair at scale. Empirical Software Engineering, 26(2), Article 20.
193. Ying, Z., Bourgeois, D., You, J., Zitnik, M., & Leskovec, J. (2019). GNNExplainer: Generating explanations for graph neural networks. In Advances in Neural Information Processing Systems 32 (NeurIPS 2019).
194. Yusuf, H. D., & Abdulrazaq, K. (2018). A conceptual framework for strengthening diagnostic quality systems in resource-limited clinical laboratories. Iconic Research and Engineering Journals, 1(11), 145-179. https://doi.org/10.64388/IREV1I11-1723090
195. Yusuf, H. D., Abdulrazaq, K., Osinem, H. E., & Eribenne, F. (2020). Advances in antimicrobial resistance surveillance using multi-laboratory national reporting networks. International Journal of Multidisciplinary Evolutionary Research, 1(2), 176-194. https://doi.org/10.54660/IJMER.2020.1.2.176-194
196. Zeng, Z., Zhang, Y., Zhang, H., & Zhang, L. (2021). Deep just-in-time defect prediction: How far are we? In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2021) (pp. 427-438).

197. Zheng, Y., Pujar, S., Lewis, B., Buratti, L., Epstein, E., Yang, B., Laredo, J., Morari, A., & Su, Z. (2021). D2A: A dataset built for AI-based vulnerability detection methods using differential analysis. In Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2021) (pp. 111-120).
198. Zhou, Y., Liu, S., Siow, J., Du, X., & Liu, Y. (2019). Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In Advances in Neural Information Processing Systems 32 (NeurIPS 2019).
199. Zhu, Q., Sun, Z., Xiao, Y., Zhang, W., Yuan, K., Xiong, Y., & Zhang, L. (2021). A syntax-guided edit decoder for neural program repair. In Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021) (pp. 341-353).
200. Zimmermann, M., Staicu, C.-A., Tenny, C., & Pradel, M. (2019). Small world with high risks: A study of security threats in the npm ecosystem. In Proceedings of the 28th USENIX Security Symposium (pp. 995-1010).