Test-Driven Development and Object-Oriented Design

Empirical Evidence vs. Practitioner Folklore

Isaac Tillema
DePaul University

ABSTRACT

Test-Driven Development (TDD) is widely regarded, not just as a testing discipline, but as a primary driver of superior object oriented (OO) design, supposedly yielding smaller classes, looser coupling, and higher cohesion. However, a significant gap exists between practitioner (engineers) folklore and empirical evidence. This report systematically investigates TDD's structural impact utilizing Chidamber and Kemerer (CK) metrics and process dissections. Empirical analysis reveals that TDD's architectural benefits are ambiguous and frequently negative; while it reduces individual class size, it often degrades cohesion and inflates explicit coupling through dependency injection. Furthermore, studies show the hallmark "test-first" sequence is statistically irrelevant to design quality. Instead, TDD's benefits stem from process granularity and uniformity. Ultimately, TDD reliably reduces pre-release defect density by 40% to 90% at the cost of a 15% to 35% productivity drop, acting as a strict forcing function for testability rather than automatically generating optimal or guaranteed OO architecture.

CCS CONCEPTS

• Empirical software validation • Software design engineering • Software quality • Object oriented architectures • Object oriented programming

KEYWORDS

Test-Driven Development (TDD), Object-Oriented Design (OOD), Chidamber and Kemerer (CK) Metrics, Software Engineering, Software Quality, Coupling and Cohesion, Dependency Injection, SOLID Principles, Static Analysis

1 INTRODUCTION

Test-Driven Development (TDD) has fundamentally inverted the traditional software development lifecycle.[1] Instead of writing production code followed by test suites, developers follow a strict micro-iterative cycle: write a failing test (Red), write the minimum code to pass it (Green), and restructure the code (Refactor), known to engineers at the Red-Green-Refactor cycle.[2] Over the years, influential engineers have promoted TDD as an emergent Object Oriented Design (OOD) methodology, arguing that testing friction actively forces smaller classes, looser coupling, and higher cohesion.[1]

However, this claim largely rests on subjective anecdote rather than empirical evidence.[6] When researchers apply quantitative static analysis to TDD codebases using established metrics (e.g., CBO, LCOM, WMC, DIT), the results are remarkably mixed and often contradict engineers' advocacy.[8] Because TDD adoption carries real costs in training and initial velocity drops, organizations need to know if these design benefits are genuine.[11]

This research bridges the gap between practitioner advocacy and empirical reality, guided by three core questions:

1. **RQ1**: Does TDD empirically lead to measurably better OO design quality (coupling, cohesion, class size, inheritance) compared to test-last or no-test approaches?
2. **RQ2**: Which specific design improvements attributed to TDD (e.g., dependency injection, interface use) are empirically supported, and which are folklore?
3. **RQ3**: To what extent do contextual factors (developer experience, process granularity) moderate the relationship between TDD and OOP design quality?

2 BACKGROUND & METRICS

In software engineering, subjective observations often codify into a phenomenon termed the "Leprechauns of Software Engineering."[6] TDD's emergent design theory relies on this. It assumes that because highly coupled, monolithic code is painful to test, the developer will naturally break it apart, resulting in optimal modularity.[3]

To evaluate this objectively, researchers use the Chidamber and Kemerer (CK) Metrics Suite:[14]

- **WMC (Weighted Methods per Class)**: Indicates overall class size. (TDD claim: Lower WMC).
- **CBO (Coupling Between Objects)**: Distinct non-inheritance classes a given class is coupled to. (TDD claim: Lower CBO).
- **LCOM (Lack of Cohesion in Methods)**: Disparity between methods sharing instance variables.

Lower LCOM = higher cohesion. (TDD claim: Lower LCOM).

- **DIT (Depth of Inheritance Tree)**: Length of class hierarchy. (TDD claim: Lower DIT).

3 RESEARCH METHODOLOGY

This research synthesizes empirical literature collection from over the past two decades, utilizing NotebookLM as a research aggregation tool. The collection includes Systematic Literature Reviews (SLRs), controlled academic/industrial quasi-experiments comparing TDD against Test-Last Development (TLD), and advanced process dissection studies utilizing biometric and chronological tracking.[9] A strict delineation is maintained between external quality (functional correctness/defects) and internal quality (structural OO metrics).[19]

4 RESULTS & ANALYSIS

Results and analysis of the following topics:

1. **RQ1**: Does TDD empirically lead to measurably better OO design quality (coupling, cohesion, class size, inheritance) compared to test-last or no-test approaches?
2. **RQ2**: Which specific design improvements attributed to TDD (e.g., dependency injection, interface use) are empirically supported, and which are folklore?
3. **RQ3**: To what extent do contextual factors (developer experience, process granularity) moderate the relationship between TDD and OOP design quality?

4.1 Empirical Evaluation of OO Design Metrics

The foundational claim that TDD inherently produces superior OO structures is heavily challenged by static analysis.

Class Complexity (WMC)

TDD does successfully reduce individual class size and complexity. Metrics such as WMC and Lines of Code (LOC) per method show consistent reductions, as writing tests for massive classes is cognitively burdensome.[21] However, overall system complexity often merely shifts from intra-class to inter-class complexity.[15]

The Paradox of Coupling (CBO)

TDD folklore insists coupling must decrease.[1] Empirically, TDD often increases coupling metrics. Janzen and Saiedian found that the Information Flow (IF) metric and the average number of method parameters (PAR) were significantly higher in test-first projects.[8] To make a class testable, developers must inject dependencies (objects) via constructors, which physically increases parameter counts and explicit structural coupling.[23]

The Degradation of Cohesion (LCOM)

Engineering folklore relies heavily on TDD organically fostering high cohesion.[1] However, studies using the LCOM metric show that TDD code is frequently significantly less cohesive than TLD code.[9] This degradation is caused by testing artifacts. Developers often expose internal states through accessor methods (getters/setters) purely for state verification in tests, which artificially inflates the LCOM score and destroys natural domain abstractions.[8]

Inheritance (DIT)

Empirical evaluations show no statistically significant differences in DIT or inheritance structures between TDD and TLD.[9]

Metric	Folklore Claim	Empirical Reality	Reason for Discrepancy
WMC	Smaller classes	**Supported**	Testing friction discourages monoliths.[21]
CBO	Lower coupling	**Contradicted**	DI explicitly increases parameter counts.[8]
LCOM	Higher cohesion	**Contradicted**	Exposed accessors (getters) degrade scores.[9]
DIT	Shallower trees	**No Difference**	Domain dictates inheritance, not testing.[9]

4.2 Micro-Architectural Patterns

Dependency Injection

The most verifiable micro-architectural benefit of TDD is the pervasive adoption of dependency injection (DI). TDD forces developers to bypass volatile dependencies (e.g., databases) to run tests quickly, naturally enforcing the Interface Segregation and Dependency Inversion principles.[23]

SOLID Principles & Encapsulation

Does TDD guarantee SOLID adherence? Controlled experiments show a correlation between SOLID compliance and TDD, but only for highly experienced developers.[28] TDD highlights design flaws through testing friction, but it does not supply the architectural vocabulary to fix them. Novices who lack deep OO knowledge merely power through the friction, writing fragile tests for poorly designed code.[29] Furthermore, novices frequently default to "state verification" by spamming getter methods, breaking encapsulation and transforming behavior rich objects into mere data structures.[31] Most advanced or experienced engineers use "Tell, Don't Ask" behavior verification via mocks, but this heavily fragments the system logic.[33]

4.3 Moderating Contextual Factors

If TDD's structural impacts are mixed, moderating variables dictate the outcome. Foundational process dissection studies by Fucci et al. deconstructed the TDD cycle into four dimensions, yielding findings that debunk core agile beliefs.[18]

1. **Sequencing (Irrelevant)**: The order in which test and production code are written (Test-First vs. Test-Last) has no statistically significant impact on quality or productivity. The "test-first" dynamic itself does not drive the benefits.[18]
2. **Granularity & Uniformity (Primary Drivers)**: Improvements in software quality heavily correlate with working in highly granular (5-10 minute) and uniform cycles. TDD improves focus by acting as a cognitive pacing mechanism, inducing a "flow state," regardless of whether the test was written first or last.[18]
3. **Developer Experience**: TDD multiplies existing architectural skill. Novices actually produce better internal code quality with iterative test-last development, as TDD's cognitive load and mocking abstractions overwhelm their lack of OO foundations.[9]

5 EXTERNAL QUALITY VS PRODUCTIVITY

If TDD does not guarantee structural OO design, its massive adoption is justified by functional correctness. Meta-analyses of industrial case studies consistently prove that TDD reduces pre-release defect density by an astonishing 40% to 90%.[37] The methodology creates a solidifying regression harness. However, this extracts a heavy toll. An empirically validated 15% to 35% decrease in developer productivity and initial velocity due to the steep overhead of test generation and mocking setup.[16]

6 CONCLUSION

This research set out to determine whether TDD mechanically produces superior or guaranteed OO design. Empirical evidence decisively concludes that it does not. Static analysis utilizing CK metrics reveals that TDD frequently yields lower cohesion, artificially inflated coupling through parameter passing, and fragmented class structures.[8] Furthermore, tracking and analysis proves the foundational dogma of "test-first" sequencing is irrelevant to design quality; instead, TDD's true cognitive benefit is enforcing fine-grained, uniform development cycles.[18]

TDD is not an automated architecture generator; it is an uncompromising auditor of pre-existing design skills. For professionals and educators, adopting TDD guarantees a massive reduction in defect density at the cost of initial velocity.[37] However, expecting TDD to organically cure poor object-oriented design is a dangerous reliance on folklore. Without concurrent, rigorous training in SOLID principles and encapsulation, TDD will merely produce poorly designed, tightly coupled systems with excellent test coverage.

ACKNOWLEDGMENTS

Vahid Alizadeh, DePaul University

REFERENCES

[1] Wikipedia. n.d. Test-driven development. Retrieved June 1, 2026 from https://en.wikipedia.org/wiki/Test-driven_development.

[2] arXiv. n.d. A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last? Retrieved June 1, 2026 from https://arxiv.org/pdf/1611.05994.

[3] Medium. n.d. A Guide to Test-Driven Development (TDD) with Real-World Examples. Retrieved June 1, 2026 from https://medium.com/@dees3g/a-guide-to-test-driven-development-tdd-with-real-world-examples-d92f7c801607.

[4] Boldare. n.d. What is Test-Driven Development? TDD Benefits & Examples. Retrieved June 1, 2026 from

https://www.boldare.com/blog/test-driven-development-tdd-definition-benefits/.

[5] Ranorex. n.d. Mocking and Dependency Injection: TDD is Hardest Problems. Retrieved June 1, 2026 from https://www.ranorex.com/blog/tdds-hardest-problems/.

[6] DOKUMEN.PUB. n.d. The Leprechauns of Software Engineering: How folklore turns into fact and what to do about it (1st ed.). Retrieved June 1, 2026 from https://dokumen.pub/the-leprechauns-of-software-engineering-how-folklore-turns-into-fact-and-what-to-do-about-it-1nbsped.html.

[7] Increment: Teams. n.d. The epistemology of software quality. Retrieved June 1, 2026 from https://increment.com/teams/the-epistemology-of-software-quality/.

[8] Digital Commons @ Cal Poly. n.d. Does Test-Driven Development Really Improve Software Design Quality? Retrieved June 1, 2026 from https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1027&context=csse_fac.

[9] M. Siniaalto and Ab. n.d. A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage. arXiv. Retrieved June 1, 2026 from https://arxiv.org/pdf/1711.05082.

[10] n.d. [PDF] Does Test-Driven Development Really Improve Software Design Quality? Retrieved June 1, 2026 from https://www.semanticscholar.org/paper/fe7117d75c48cffe0a8e1d04e595d865b60ab65d.

[11] Diva-Portal.org. n.d. Test-Driven Development. Retrieved June 1, 2026 from https://www.diva-portal.org/smash/get/diva2:1971520/FULLTEXT01.pdf.

[12] n.d. Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review. Retrieved June 1, 2026 from https://ieeexplore.ieee.org/document/5770623/.

[13] n.d. Test-Driven Development (TDD) - What it is and How to Implement it. Retrieved June 1, 2026 from https://amplitude.com/explore/experiment/what-is-test-driven-development.

[14] n.d. Unit Test Case Design Metrics in Test Driven Development. Retrieved June 1, 2026 from http://article.sapub.org/10.5923.j.se.20120203.01.html.

[15] Shyam R. Chidamber and Chris F. Kemerer. n.d. A METRICS SUITE FOR OBJECT ORIENTED DESIGN. M.I.T. Sloan School of Management / ESO.org. Retrieved June 1, 2026 from https://www.eso.org/~tcsmgr/oowg-forum/TechMeetings/Articles/OOMetrics.pdf.

[16] ResearchGate. n.d. The Effects of Test Driven Development on Internal Quality, External Quality and Productivity: A systematic review. Retrieved June 1, 2026 from https://www.researchgate.net/publication/295863661_The_Effects_of_Test_Driven_Development_on_Internal_Quality_External_Quality_and_Productivity_A_systematic_review.

[17] Microsoft Research. n.d. Realizing quality improvement through test driven development: results and experiences of four industrial teams. Retrieved June 1, 2026 from https://www.microsoft.com/en-us/research/wp-content/uploads/2009/10/Realizing-Quality-Improvement-Through-Test-Driven-Development-Results-and-Experiences-of-Four-Industrial-Teams-nagappan_tdd.pdf.

[18] arXiv. n.d. A Dissection of the Test-Driven Development Process: Does ... Retrieved June 1, 2026 from https://arxiv.org/abs/1611.05994.

[19] ResearchGate. n.d. When, How, and Why Developers (Do Not) Test in Their IDEs. Retrieved June 1, 2026 from https://www.researchgate.net/publication/279037955_When_How_and_Why_Developers_Do_Not_Test_in_Their_IDEs.

[20] IEEE Computer Society. n.d. A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last? Retrieved June 1, 2026 from https://www.computer.org/csdl/journal/ts/2017/07/07592412/13rRUB6Sq2j.

[21] InfoWorld. n.d. Test-Driven Development and Software Quality. Retrieved June 1, 2026 from https://www.infoworld.com/article/2158976/test-driven-development-and-software-quality.html.

[22] n.d. An Empirical Evaluation of the Impact of Test-Driven Development on Software Quality. Retrieved June 1, 2026 from http://users.csc.calpoly.edu/~djanzen/pubs/dissertation.pdf.

[23] Nuno Sousa. n.d. TDD and Dependency Injection. How Two simple techniques can make a… Medium. Retrieved June 1, 2026 from https://medium.com/@nuno.mt.sousa/tdd-and-dependency-injection-35d4cd8a28be.

[24] n.d. SOLID design principles make test-driven development (TDD) faster and easier. Retrieved June 1, 2026 from https://gkovan.medium.com/solid-design-principles-makes-test-driven-development-faster-and-easier-35c9eec22ff1.

[25] Beningo Embedded Group. n.d. 9 Software Architecture Metrics for Sniffing Out Issues. Retrieved June 1, 2026 from https://www.beningo.com/9-software-architecture-metrics-for-sniffing-out-issues/.

[26] Codurance. n.d. Getters and Setters Considered Harmful. Retrieved June 1, 2026 from https://www.codurance.com/publications/2018/03/20/getters-and-setters-considered-harmful.

[27] ResearchGate. n.d. Does Test-Driven Development Improve the Program Code? Alarming Results from a Comparative Case Study. Retrieved June 1, 2026 from https://www.researchgate.net/publication/221200732_Does_Test-Driven_Development_Improve_the_Program_Code_Alarming_Results_from_a_Comparative_Case_Study.

[28] Coding Is Like Cooking. n.d. SOLID principles and TDD. Retrieved June 1, 2026 from https://coding-is-like-cooking.info/2012/09/solid-principles-and-tdd/.

[29] Reddit. n.d. TDD, Where Did It All Go Wrong : r/programming. Retrieved June 1, 2026 from https://www.reddit.com/r/programming/comments/oum69m/tdd_where_did_it_all_go_wrong/.

[30] n.d. Are there any cases when one should not use Test Driven Development? [duplicate]. Retrieved June 1, 2026 from https://softwareengineering.stackexchange.com/questions/234753/are-there-any-cases-when-one-should-not-use-test-driven-development.

[31] Paweł Pluta. n.d. Tell, Don't Ask — Learn to Talk to Your Objects. TechTalks@Vattenfall. Retrieved June 1, 2026 from https://medium.com/vattenfall-tech/tell-dont-ask-learn-to-talk-to-your-objects-45d7c4aa61fe.

[32] Nikos Voulgaris. n.d. How getters and setters harm encapsulation. Retrieved June 1, 2026 from https://nvoulgaris.com/how-getters-and-setters-harm-encapsulation/.

[33] Martin Fowler. n.d. Tell Dont Ask. Retrieved June 1, 2026 from https://martinfowler.com/bliki/TellDontAsk.html.

[34] Clean Coder. n.d. TDD in Clojure. Retrieved June 1, 2026 from https://sites.google.com/site/unclebobconsultingllc/uncle-bob-consulting-llc/articles/tdd-in-clojure.

[35] The Morning Paper. n.d. A dissection of the test-driven development process: does it really matter to test-first or test-last? Retrieved June 1, 2026 from https://blog.acolyer.org/2017/06/13/a-dissection-of-the-test-driven-development-process-does-it-really-matter-to-test-first-or-test-last/.

[36] DCC/UFMG. n.d. Test-Driven Development Benefits Beyond Design Quality: Flow State and Developer Experience. Retrieved June 1, 2026 from https://www.dcc.ufmg.br/~pcalais/papers/tdd-flow-icse-nier-2023.pdf.

[37] n.d. Has test driven development (TDD) actually benefited a real world project? Retrieved June 1, 2026 from https://softwareengineering.stackexchange.com/questions/148329/has-test-driven-development-tdd-actually-benefited-a-real-world-project.

[38] ramblinations. n.d. Hack Better, with SCIENCE. Retrieved June 1, 2026 from https://ramblinations.com/hack-better-with-science/.

[39] Matheus Marabesi. n.d. TDD. Retrieved June 1, 2026 from https://marabesi.com/tdd.

[40] ResearchGate. n.d. A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last? | Request PDF. Retrieved June 1, 2026 from https://www.researchgate.net/publication/386802630_A_Dissection_of_the_Test-Driven_Development_Process_Does_It_Really_Matter_to_Test-First_or_to_Test-Last.

[41] arXiv. n.d. Why Research on Test-Driven Development is Inconclusive? Retrieved June 1, 2026 from https://arxiv.org/pdf/2007.09863.