ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
University of the Peloponnese

ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
Π.Μ.Σ. ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

A SOLID Grasp of C# Design: Principles, Refactoring, and a Case-Driven Approach

Angeliki Freskou

Reg. no.: 2022201802032

Supervisor: Costas Vasilakis

Tripolis, October 2025

Acknowledgements

This work could not have been completed without the guidance and support of many people. Sincere thanks are extended to **Professor Vassilakis** for his confidence in this endeavor and for the clear direction and constructive criticism that shaped the research into its final form. His trust in the project and steady mentorship were invaluable at every stage.

Appreciation is also expressed to the **staff of the Department of Informatics and Telecommunications at the University of the Peloponnese** for the knowledge, resources, and academic environment that enabled steady growth as an engineer and researcher.

Finally, heartfelt thanks go to **family and friends** for their patience, motivation, and encouragement throughout this journey. In particular, deepest gratitude is owed to **my husband, Grigoris Dimitroulakos**, whose unwavering support and understanding sustained me during the most demanding periods. His belief in me provided constant strength and renewed determination to reach this goal.

Abstract

The thesis investigates how the combined application of the SOLID principles and the GRASP patterns can be used to produce robust C# software designs that remain maintainable, testable, and extensible as systems evolve. A unified, principle-to-practice framework is presented in which SOLID secures class- and component-level integrity while GRASP structures responsibilities and collaborations at the architectural level. The approach is operationalized in the .NET ecosystem through idiomatic refactorings—interface extraction, role-focused interface segregation, strategy-based composition, and dependency inversion realized via dependency injection—so that variation can be absorbed at stable seams owned by high-level policy. To move beyond stylistic guidance, the work employs metric-guided evaluation and a cohesive C# case study representative of an e-commerce workflow. Changes are validated using static and change-based indicators (coupling/cohesion proxies, instability indices, files-touched per feature), along with test metrics (coverage and mutation score). The results indicate that dependency boundaries defined and verified at the policy layer localize change, reduce “shotgun surgery,” and enable faster, more reliable testing through mocks and contract tests; extension by addition is favored over risky edits, and architectural refactoring becomes safer. Contributions include: (i) a mapped correspondence between SOLID and GRASP that clarifies when and how to apply each; (ii) a refactoring playbook tied to expected metric movement; (iii) case studies demonstrating end-to-end impact in C#; and (iv) pedagogical scaffolding suitable for studio-style instruction. Limitations are noted regarding external validity across real-time or resource-constrained domains and the use of proxy metrics for maintainability. Future work is outlined for principle-aware analyzers, boundary verification in continuous integration, broader multi-team replication, and curriculum-ready case libraries, with the aim of turning design principles into repeatable, measurable engineering practice.

Περίληψη

Η παρούσα διπλωματική εργασία διερευνά πώς ο συνδυασμός των αρχών SOLID και των προτύπων GRASP μπορεί να χρησιμοποιηθεί για την παραγωγή ανθεκτικών σχεδίων λογισμικού σε C#, τα οποία παραμένουν συντηρήσιμα, ελέγξιμα και επεκτάσιμα καθώς τα συστήματα εξελίσσονται. Παρουσιάζεται ένα ενοποιημένο πλαίσιο από τη θεωρία στην πράξη, στο οποίο οι αρχές SOLID διασφαλίζουν την ακεραιότητα σε επίπεδο κλάσης και συνιστώσας, ενώ τα πρότυπα GRASP οργανώνουν τις ευθύνες και τις συνεργασίες σε αρχιτεκτονικό επίπεδο.

Η προσέγγιση υλοποιείται στο οικοσύστημα .NET μέσω ιδιοτυπικών αναδομήσεων (refactorings), όπως εξαγωγή διεπαφών, διαχωρισμός διεπαφών βάσει ρόλων, σύνθεση μέσω προτύπων στρατηγικής και αντιστροφή εξάρτησης μέσω dependency injection, έτσι ώστε η ποικιλία να απορροφάται σε σταθερά σημεία ελέγχου που ανήκουν στην ανώτερη πολιτική του συστήματος.

Για να υπερβεί το επίπεδο της απλής στυλιστικής καθοδήγησης, η εργασία εφαρμόζει αξιολόγηση με βάση μετρικές και παρουσιάζει μια συνοχή μελέτη περίπτωσης σε C#, αντιπροσωπευτική ενός ηλεκτρονικού εμπορικού σεναρίου (e-commerce workflow). Οι αλλαγές επικυρώνονται μέσω στατικών και δυναμικών δεικτών (πληρεξούσιες μετρικές σύζευξης/συνοχής, δείκτες αστάθειας, αριθμός αρχείων που τροποποιούνται ανά λειτουργία), καθώς και μετρικών δοκιμών (κάλυψη κώδικα και βαθμός μετάλλαξης).

Τα αποτελέσματα δείχνουν ότι τα όρια εξάρτησης, όπως ορίζονται και επαληθεύονται στο επίπεδο πολιτικής, τοπικοποιούν τις αλλαγές, μειώνουν το φαινόμενο της “χειρουργικής με καραμπίνα” (*shotgun surgery*) και επιτρέπουν ταχύτερο και πιο αξιόπιστο έλεγχο μέσω mocks και δοκιμών συμβολαίου (contract tests)· η επέκταση μέσω προσθήκης προτιμάται έναντι επικίνδυνων τροποποιήσεων, και η αρχιτεκτονική αναδόμηση καθίσταται ασφαλέστερη.

Οι συνεισφορές περιλαμβάνουν:

(i) έναν αντιστοιχισμένο χάρτη μεταξύ SOLID και GRASP που αποσαφηνίζει πότε και πώς εφαρμόζεται κάθε αρχή·
(ii) ένα εγχειρίδιο αναδομήσεων συνδεδεμένο με την αναμενόμενη μεταβολή μετρικών
(iii) μελέτες περίπτωσης που αποδεικνύουν τον ολιστικό αντίκτυπο σε C#·και

(iv) διδακτικό υλικό κατάλληλο για διδασκαλία τύπου εργαστηρίου (*studio-style instruction*).

Αναγνωρίζονται περιορισμοί όσον αφορά την εξωτερική εγκυρότητα σε τομείς πραγματικού χρόνου ή με περιορισμένους πόρους, καθώς και στη χρήση έμμεσων μετρικών για τη συντηρησιμότητα. Τέλος, προτείνεται μελλοντική εργασία για αναλυτές ευαισθητοποιημένους στις αρχές σχεδίασης, επαλήθευση ορίων σε συνεχή ολοκλήρωση (*continuous integration*), ευρύτερη επαναληψιμότητα σε πολυομαδικά περιβάλλοντα και εκπαιδευτικές βιβλιοθήκες μελετών περίπτωσης, με στόχο τη μετατροπή των αρχών σχεδίασης σε επαναλήψιμη, μετρήσιμη μηχανική πρακτική.

Table of contents

Table of Figures

Index of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Modern software systems are embedded in nearly every facet of contemporary society. From e-commerce platforms powering global trade to mobile applications shaping individual daily habits, and from mission-critical enterprise resource planning (ERP) systems to the cloud-native services that orchestrate international communication, software today serves as the foundational infrastructure of economic and social activity. This ubiquity, however, is accompanied by an inherent complexity: as systems evolve to meet rapidly shifting requirements, developers are confronted with challenges of scale, maintainability, and long-term adaptability.

The central paradox of software engineering is that software must both **change continuously** and **remain stable**. Customers and organizations demand new features, integrations, and compliance with shifting legal or security landscapes. Yet at the same time, stakeholders expect that existing functionality—already validated and deployed—will remain dependable. The tension between **changeability** and **reliability** is the enduring challenge of software design. Without clear architectural guidance, codebases often devolve into brittle collections of patches. Technical debt accumulates; onboarding of new team members slows; testing costs rise; and, ultimately, organizations lose confidence in their ability to evolve their systems safely.

To counteract this tendency, the software engineering community has, over decades, distilled experiential wisdom into **design principles**. These principles are not recipes or rigid blueprints, but rather heuristics and guidelines that capture recurring insights about what makes code resilient to change. Among the most widely recognized families of such principles are **SOLID** and **GRASP**.

- **The SOLID principles**, popularized by Robert C. Martin in the early 2000s, crystallize five core guidelines for object-oriented software:
    - **Single Responsibility Principle (SRP):** a class should have only one reason to change.

    - **Open/Closed Principle (OCP):** entities should be open for extension but closed for modification.
    - **Liskov Substitution Principle (LSP):** subtypes must be substitutable for their base types.
    - **Interface Segregation Principle (ISP):** clients should not be forced to depend on methods they do not use.
    - **Dependency Inversion Principle (DIP):** high-level modules should depend on abstractions, not on details.
- **The GRASP patterns** (General Responsibility Assignment Software Patterns), introduced by Craig Larman in the late 1990s, complement SOLID by offering heuristics for assigning responsibilities in a way that balances cohesion and coupling across the system: Information Expert, Creator, Controller, Low Coupling, High Cohesion, Polymorphism, Indirection, Pure Fabrication, and Protected Variations.

While SOLID emphasizes **class-level design integrity**, GRASP emphasizes **responsibility assignment at the system level**. Taken together, they provide a complementary methodology: SOLID ensures that individual modules remain cohesive and safe to evolve, while GRASP ensures that the interactions between modules remain intelligible, decoupled, and balanced.

The motivation for combining these two families is twofold. First, software teams in industry frequently struggle when principles are applied in isolation. A system that is "SOLID-compliant" in terms of class design may still suffer from poorly distributed responsibilities, while a GRASP-compliant responsibility assignment may falter if the resulting classes do not respect SRP, OCP, or LSP. Second, the empirical evidence from both academia and industry suggests that systems designed with attention to **both modularity and responsibility assignment** exhibit measurably better maintainability, testability, and evolvability. This dual perspective therefore offers practitioners not only theoretical elegance but also pragmatic resilience.

The context of this thesis is the **C# and .NET ecosystem**, which provides an ideal arena for exploring these principles. C# offers rich support for object orientation, generics, interfaces, reflection, and language-integrated query (LINQ), while the .NET ecosystem provides powerful frameworks for dependency injection, test automation, and architectural layering. Furthermore, .NET is widely used in enterprise settings, where long-lived systems must evolve safely over years

or even decades. By grounding the discussion in idiomatic C# examples, this work ensures that the principles are not abstract slogans but are concretely applicable to modern industrial practice.

Finally, the relevance of this work extends beyond industry. In the **academic domain**, teaching design principles is essential for cultivating students' ability to think critically about architecture rather than only about coding syntax. Experience shows that graduates who can reason about maintainability, testability, and extensibility are better prepared for professional practice. By providing C#-based demonstrations of SOLID and GRASP in action, this thesis aims to contribute also to pedagogy, offering students concrete bridges between design theory and coding practice.

1.2 Research Problem and Rationale

Although both SOLID and GRASP have been widely taught and cited in literature, their combined application has rarely been systematized in a way that provides actionable guidance. Instead, practitioners often encounter fragmented advice: tutorials on SOLID without reference to responsibility assignment, or GRASP discussions without consideration of how to enforce design contracts. This fragmentation leads to two problems.

First, teams may implement principles superficially. For example, a developer might create multiple interfaces to "satisfy" ISP but fail to map those interfaces to actual responsibilities in the domain, creating artificial fragmentation without genuine decoupling. Similarly, a team may adopt GRASP's Information Expert but overlook that the resulting class has accumulated too many reasons to change, violating SRP. Without integration, principles risk being applied mechanically rather than thoughtfully.

Second, tooling support remains asymmetric. Static analysis tools such as SonarQube or NDepend can flag probable SOLID violations—large classes, deep inheritance hierarchies, unused interface members—but there is little automated support for detecting GRASP misapplications. This gap can leave responsibility assignment largely in the realm of subjective judgement. A unified framework that clarifies **how SOLID and GRASP interrelate** could therefore empower both better human decision-making and more targeted tool support.

The rationale of this thesis is that by **mapping each SOLID principle to relevant GRASP patterns**, a systematic methodology emerges. This methodology provides not only diagnostic power—helping teams recognize when a design is drifting—but also prescriptive guidance,

suggesting concrete refactoring strategies grounded in both families of principles. By embedding these mappings in C# examples, the thesis demonstrates how violations can be recognized, how refactorings can be conducted incrementally, and how the resulting design can be explained both in theoretical and practical terms.

1.3 Research Objectives

The objectives of this thesis can be articulated across three complementary dimensions: theoretical, analytical, and practical.

1.3.1 Theoretical Objectives

1. **Synthesize foundations** by surveying the origins and rationales of both SOLID and GRASP, highlighting their conceptual alignments and differences.
2. **Clarify design intent** by examining how principles have been historically interpreted in both academic and industrial contexts.
3. **Extend understanding** by arguing for the synergy of the two families, showing that SOLID refines local correctness while GRASP ensures global balance.

1.3.2 Analytical Objectives

1. **Identify anti-patterns** in C# code that correspond to violations of SOLID and GRASP principles.
2. **Measure violations** using static analysis metrics such as LCOM, RFC, coupling indices, and change frequency analytics.
3. **Evaluate trade-offs** by distinguishing between objective indicators (metrics, tool warnings) and subjective factors (domain semantics, performance constraints).

1.3.3 Practical Objectives

1. **Demonstrate refactorings** by providing step-by-step C# examples that transform flawed code into principle-compliant design.
2. **Map refactorings to GRASP** so that each code change can be justified not merely in terms of syntax but also in terms of responsibility assignment.

3. **Validate through a case study** by refactoring a small e-commerce module iteratively, applying multiple principles in concert.
4. **Explore advanced principles**—Interface Segregation, Dependency Inversion, and Dependency Injection—to illustrate how they secure large-scale maintainability and testability.

By fulfilling these objectives, the thesis aims to deliver both **scholarly insight** and **practical utility**.

1.4 Contribution of the Thesis

This work contributes in several ways:

- **An integrated framework**: a systematic mapping of SOLID principles to GRASP patterns, showing how class-level correctness and system-level responsibility assignment reinforce each other.
- **Refactoring playbooks**: step-wise examples in idiomatic C#, each demonstrating how to resolve specific violations and how to justify the changes with design principles.
- **Metric-guided analysis**: demonstrations of how code metrics and version-control analytics can be used to diagnose violations objectively, while still allowing for subjective judgement.
- **Case study evidence**: a realistic e-commerce module that demonstrates the iterative application of principles, making the benefits concrete in terms of reduced complexity, improved testability, and enhanced extensibility.
- **Pedagogical value**: an accessible yet rigorous treatment of design principles tailored for final-year Informatics students and practitioners alike.

1.5 Thesis Structure

The remainder of the thesis is organized to progress from foundations to principle-specific analyses and, finally, to synthesis and outlook.

Chapter 2 presents the literature review, where the SOLID principles and the GRASP patterns are surveyed and their emerging synergy across academic, educational, and industrial settings is summarized. Attention is drawn to the differing levels of tool support and to the role of empirical evidence in motivating an integrated stance.

Chapter 3 establishes the **foundational mechanisms behind dependencies**, which serve as the connective tissue of object-oriented systems. Dependencies are examined along four orthogonal axes so that later principle chapters can refer to a common vocabulary and set of trade-offs. First, the **type of dependency** is catalogued (inheritance/generalization, association/aggregation, composition, delegation, parameter-level/usage-only, and dynamic/reflective links). Second, the **abstractness of the dependency reference** is analyzed (concrete classes, abstract classes, interfaces, delegates/callbacks, and dynamic typing or reflection), emphasizing how higher abstraction levels alter substitutability and coupling. Third, the **creation timing and instantiation mechanisms** are compared (direct `new`, constructor injection, factory and static factory methods, service locator, container-based binding/configuration, and reflection-based activation), with attention to how each choice shifts binding time, flexibility, and operational risk. Fourth, the **span and depth of dependency chains** are characterized (horizontal collaborations among peers versus vertical call chains across layers), together with their implications for modularity, cohesion, and SRP. A unifying synthesis closes the chapter by relating these axes to GRASP (Low Coupling, Indirection, Protected Variations) and to SOLID (especially OCP and DIP), so that subsequent chapters can ground refactorings in dependency mechanics rather than slogans.

Chapter 4 addresses the **Single Responsibility Principle (SRP)**, relating responsibility cohesion to GRASP's Information Expert and Pure Fabrication, and balancing objective indicators (e.g., cohesion and usage metrics) with necessary semantic judgement; a practical workflow for evidence-guided cohesion refactoring is provided.

Chapter 5 develops the **Open/Closed Principle (OCP)**, mapping openness to GRASP strategies (Protected Variations, Polymorphism, Indirection, Pure Fabrication), pairing change-proneness and instability measures with a refactoring playbook that realizes extension by addition rather than risky edits.

Chapter 6 treats the **Liskov Substitution Principle (LSP)**, focusing on behavioral contracts and substitutability hazards, and shows how precise specifications and tests preserve trust in extension hierarchies.

Chapter 7 examines the **Interface Segregation Principle (ISP)**, arguing for client-specific contracts to reduce coupling, aligning with GRASP's Low Coupling and High Cohesion, and providing a stepwise method from "fat" to focused interfaces.

Chapter 8 presents the **Dependency Inversion Principle (DIP)** and its realization via **Dependency Injection**, showing how abstractions owned by policy localize change and enable reliable testing; links to GRASP's Indirection and Protected Variations are made explicit.

Chapter 9 offers **Conclusions, Limitations, and Future Work**, synthesizing the contributions, reflecting on scope and validity, and outlining avenues for tooling, empirical replication, and pedagogy.

1.6 Significance of the Work

The significance of this thesis lies not only in its theoretical synthesis but also in its pragmatic orientation. By bridging SOLID and GRASP within a unified framework, it responds to a gap in both scholarship and practice. By grounding the discussion in C#, it ensures relevance to a major industrial ecosystem. By coupling analysis with refactoring playbooks, it transforms principles into actionable routines. Finally, by including metrics and case study validation, it demonstrates that design principles are not abstract ideals but measurable and beneficial practices.

In an era where software complexity continues to rise and development teams are under relentless pressure to deliver rapidly while maintaining quality, principles that foster **robust, evolvable, and understandable design** are not luxuries—they are necessities. This thesis therefore positions SOLID and GRASP not as competing alternatives but as complementary facets of a comprehensive design methodology that can equip students, practitioners, and researchers alike to meet the challenges of contemporary software engineering.

The remainder of the document is not available herein, due to the embarbo period specified in the thesis release form.