# Injecting Software Guarantees in Brownfield Software Development

**Students: Ishi Golub**

**Advisor: Prof. Aspen Olmsted, Ph.D.**

**School of Computing and Data Science**
**Wentworth Institute of Technology**

## Abstract

Securing a large codebase that is in production is a persistent challenge in software engineering. This project introduces a novel Logic-Driven Code Weaving Framework that elevates UML from passive documentation to an active, executable specification. The system parses OCL invariants and pre- and postconditions to derive granular rules for code modification. Unlike label-based approaches, OCL allows for the specification of sophisticated logic—such as conditional security constraints or data integrity contracts—that must be enforced across the application. To implement these directives, the framework utilizes Aspect-Oriented Programming (AOP) through Load-Time Weaving (LTW). This allows the system to intercept execution points and dynamically inject the necessary security code (e.g., validating an OCL invariant before a method executes). This approach ensures that the resulting software is "correct by construction" relative to its model, reducing the risk of manual implementation errors and architectural drift. This research offers a significant advancement in Model-Driven Development (MDD), providing a scalable pathway for automated, policy-based code enforcement in high-assurance software systems.

## Implementation

The framework implementation bridges high-level OCL specifications with the SuiteCRM (PHP) ecosystem through a three-stage automated pipeline:

1. Specification Parsing (PlantUML + OCL)Architectural models are authored in PlantUML, embedding OCL invariants within class definitions. A custom PHP Parser (leveraging nikic/php-parser) extracts these constraints, mapping UML entities to corresponding SuiteCRM Beans (e.g., Accounts, Opportunities).
2. Constraint TransformationOCL invariants are translated into executable PHP Logic. For example, a security constraint context Opportunity inv: self.amount < 10000 or self.assigned_user.role = 'Manager' is converted into a validation closure.
3. Automated Weaving (The "Action" Phase) To inject this logic without modifying SuiteCRM core files, the framework utilizes: Load-Time Weaving (LTW): Utilizing the AOP-PHP (PECL) extension, the system defines Pointcuts targeting SugarBean::save(). The translated OCL logic is woven as a "Before Advice" to validate state prior to database persistence..
4. Enforcement and FeedbackIf an OCL invariant is violated at runtime, the woven aspect intercepts the process, triggers a SugarApplication::appendErrorMessage() , and halts the transaction, ensuring the running system remains compliant with the architectural blueprint.

## Example

**1. The Design Blueprint (PlantUML + OCL)**

In the diagram, you define a formal constraint using OCL syntax. This acts as the "source of truth" for the framework.

```
@startuml
class Opportunity {
  +amount: float
  +sales_stage: string
  --
  {context Opportunity inv:
    self.sales_stage = 'Closed Won' implies self.amount > 0}
}
@enduml
```

**2. The Implementation (Woven PHP)**

The framework parses the OCL above and generates a Pointcut (where to inject) and Advice (what to execute). In SuiteCRM, we target the Opportunity bean's save() method.

```
class SecurityIntegrityAspect {

public function beforeSave(MethodInvocation $invocation) {

  $bean = $invocation->getThis();

  if ($bean->sales_stage === 'Closed Won' && $bean->amount <= 0) {
      SugarApplication::appendErrorMessage("Architectural Violation:
Closed Won deals must have an amount.");
      SugarApplication::redirect("index.php?module=Opportunities&action=Ed
itView&record={$bean->id}");

   exit;

  }

 }
}
```

## Conclusions

The implementation within **SuiteCRM** highlights the practical utility of this approach:

- **Separation of Concerns:** Cross-cutting logic (like complex security invariants) is kept entirely separate from core PHP business logic.
- **Reduced Complexity:** Developers are no longer required to manually instrument every entry point; the **AOP-PHP** layer ensures universal enforcement.
- **High Fidelity:** The use of **Load-Time Weaving** ensures that the running application remains strictly compliant with the modeled constraints, providing a robust framework for building high-assurance enterprise systems.

Ultimately, this methodology transforms UML from a static drawing into a **dynamic controller**, bridging the gap between high-level design and runtime reality.

## Bibliography

**1.A. Olmsted, Security-Driven Software Development. London, UK: PackT Press, 2024.**

**2.A. Olmsted, "Secure software development through non-functional requirements modeling," 2016 International Conference on Information Society (i-Society), Dublin, Ireland, 2016, pp. 22-27, doi: 10.1109/i-Society.2016.7854164.**