마틴 파울러 형은 뭐라 할까 ? What is the strangler pattern and how does it work?

Strangler 패턴은 소프트웨어 팀이 레거시 시스템을 점진적으로 폐기하고 대규모 재작성의 함정을 피할 수 있게 해줍니다. 이 패턴을 검토하고 관련된 단계를 자세히 살펴보겠습니다.

레거시 응용 프로그램 시스템에서의 Migration은 종종 중대한 코드 재작성 과정을 필요로 합니다. 그러나 전체 시스템을 대대적으로 개선하고 시스템을 오프라인 상태로 가져가는 대신, 기존 레거시 시스템을 점진적으로 폐기하고 동시에 새로운 기능을 점진적으로 추가하는 패턴을 구현할 수 있을 것입니다.

이 접근법은 가 Strangler 패턴으로 명명한 것으로, 일반적으로 "큰 덩어리의 똥"으로 불리는 단일 응용 프로그램 시스템을 점진적으로 업데이트하면서 여전히 운영 중인 상태로 유지합니다. 오늘은 Strangler 패턴이 무엇인지, 어떻게 구현하는지 및 사용 사례 예시와 함께 살펴보겠습니다.

What is the strangler pattern?

을 상상해보세요. 하나의 옵션은 모든 부분을 완전히 분해하고 여러 달 동안 재건축하는 것입니다. 그러나 모든 부품을 교체한 후에도 정말로 작동할 것을 얼마나 확신할 수 있을까요? 그리고 차고에 있는 동안 모터사이클을 사용하고 싶지만 사용할 수 없다면 어떻게 할까요?

입니다. 이것에는 두 가지 주요 이점이 있습니다. 첫 번째 이점은 부품을 점진적으로 교체하면 모터사이클을 여전히 사용할 수 있을 가능성이 높다는 것입니다. 완전히 재건축을 기다릴 필요가 없습니다. 두 번째 이점은 수정 또는 교체가 작동하지 않는 경우, 모든 것을 동시에 검사할 필요가 없으므로 문제를 식별하기가 훨씬 쉽다는 것입니다. 결국, 작동 중지하지 않고 업데이트된 모터사이클을 얻게 될 것입니다.

으로 작동합니다. 응용 프로그램 시스템을 완전히 분해하고 코드를 다시 작성하는 대신, 이 패턴은 개발 팀에게 시스템을 완전히 중단할 필요 없이 코드와 기능의 일부를 점진적으로 업데이트할 수 있는 방법을 제공합니다. 결국, 모든 서비스와 구성 요소가 새로운 응용 프로그램 시스템과 통합되도록 리팩터링되고 레거시 시스템은 폐기됩니다.

이렇게 하면 로 진행될 수 있습니다. 개발 팀은 두 번째 코드 베이스를 구현하는 데 너무 많은 걱정을 할 필요가 없으며, 대신 한 번에 하나의 서비스 또는 기능을 리팩터링하는 데 집중할 수 있습니다. 이것은 또한 이전 코드를 관리하는 팀과 새로운 코드를 관리하는 또 다른 팀을 만들 필요가 없다는 것을 의미합니다.

How to implement the strangler pattern

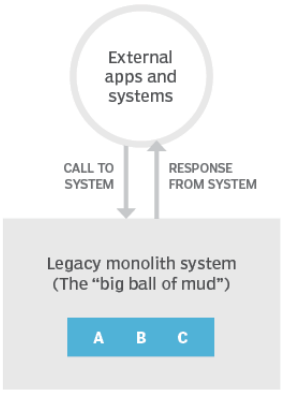
표면적으로 Strangler 패턴은 복잡해 보일 수 있습니다. 그러나 올바른 절차를 따른다면 실제로 구현하기가 매우 간단합니다.

아래 다이어그램은 Strangler 패턴 구현과 관련된 단계를 설명합니다.

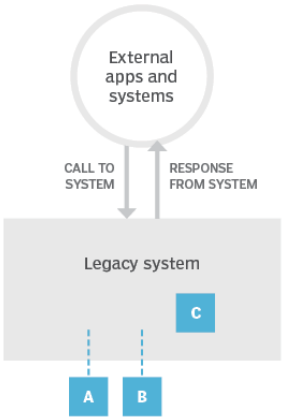
The Strangler Pattern

These diagrams illustrate the seven basic steps of implementing the Strangler Pattern to modernize a legacy application system.

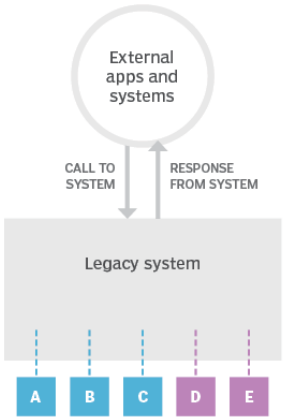
1. The system starts out as a tightly coupled "big ball of mud."



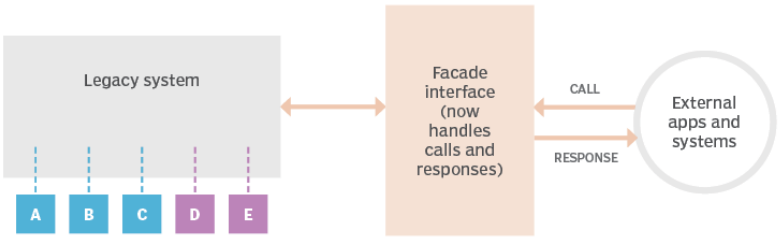
2. Begin to refactor code associated with specific service functions. Slowly turn blocks of refactored code into subsystem modules.



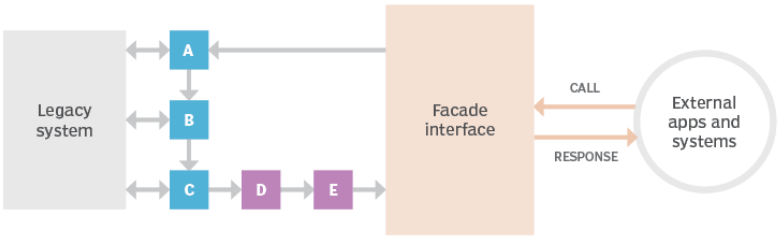
3. Add modernized features by creating new, testable subsystems that still align with existing subsystems.



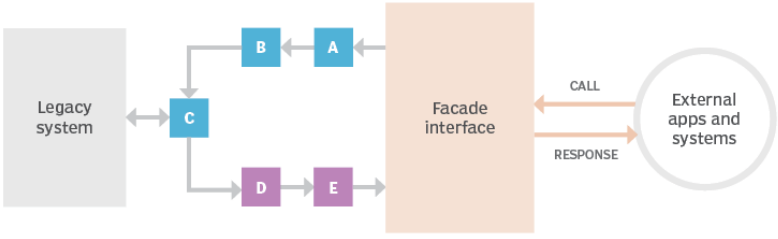
4. Create a facade interface that abstracts call and response processes from the legacy system. This interface now becomes the main point of contact for external apps and systems.



5. Remodel and isolate application flow to move solely through the refactored subsystems, while still keeping some services coupled with the legacy system.



6. Iteratively eliminate interactions between the legacy system and new subsystems until only one subsystem provides access to the legacy system.



7. Retire the legacy system, and frame a new, modernized system that acquires both the decoupled subsystems and facade interface. Application flow moves strictly through new