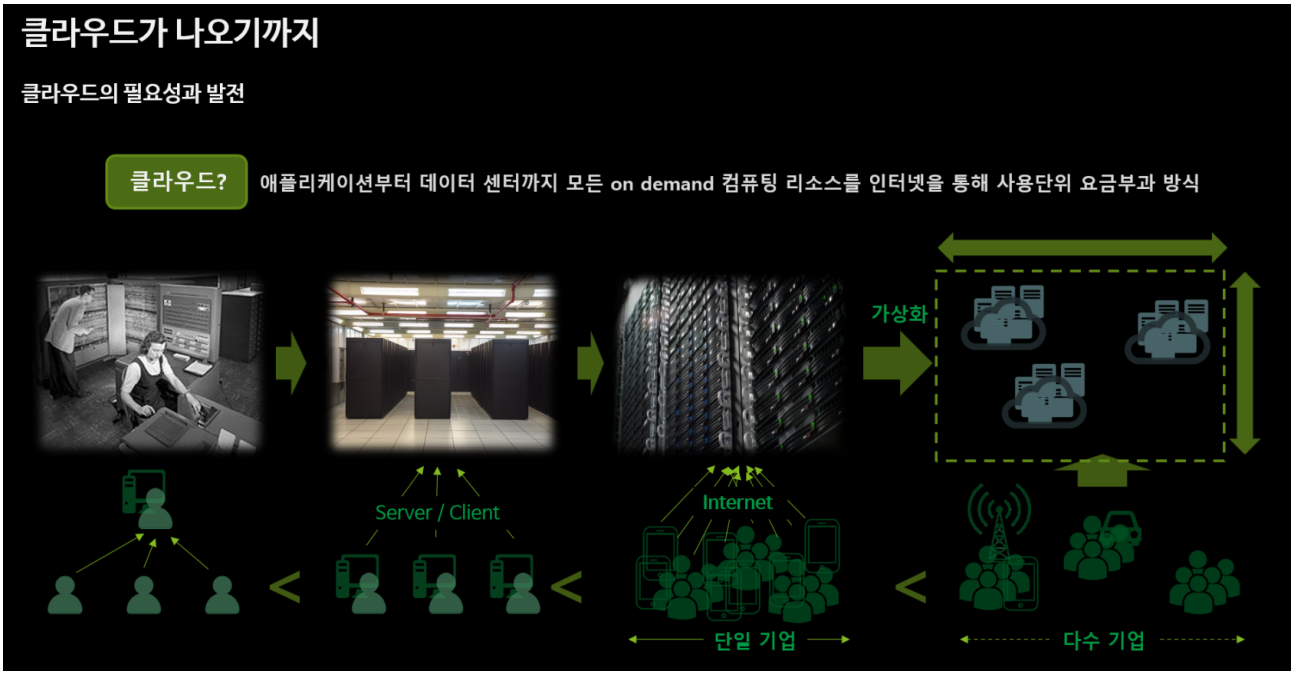
01. Why 요즘

기업의 인프라 시스템의 전환의 여정을 살펴보고 클라우드로 대표되는 클라우드네이티가 나오기까지의 기업 환경을 분석
현 시점의 차세대와 같은 기업이 가지고 있는 대규모 변화에 있어 클라우드를 기본으로 하는 기술의 변화의 적용이 필요한 이유 및 적용 방안을 제시

- 클라우드가 나오기 까지는 ?
- Big-Bang Pattern 차세대급 구축이 가능할까 ?
- 그럼 Legacy의 Modernization이 필요해, 방법은 ?
- 마틴 파울러 형은 뭐라 할까 ? What is the strangler pattern and how does it work?
- 야금야금(strangler pattern) 작전의 조건 ?
- 그럼 야금야금 자원의 추가는 ? 그런 인프라는 ?

클라우드가 나오기 까지는 ?

클라우드가 나오기까지의 시스템은 어떤 방향으로 변화하는가 ?



서비스를 중심으로 보는 시스템의 진화

· 판게아시절:

처음에는 하나의 중앙 컴퓨터가 오퍼레이터에 의하여 시분할로 작업을 순차적으로 수행하고 , 수행한 결과를 문서로 출력하여 업무에 반영하는 형태의 **중앙 집권식의 매우 폐쇄적인 시스템 (한정적인 인력이 운영하고 일부 업무에 제한적인 자동화가 이루어짐)**

· C/S 컴퓨팅:

중앙의 대용량 처리 서버를 중심으로 여러 대의 컴퓨터가 네트워크를 통해 협력하여 작업을 처리하는 방식
 이 시기를 C/S 즉 서버 클라이언트 시대를 말하며, 중앙에 서버와 각각의 단말기들이 연결되어서 서버의 컴퓨팅 파워를 사용하는 형태입니다.
중앙 집권식이긴 하였으나 기업 내부에 시스템을 공유하여 **내부 직원들에 시스템을 오픈하는 시스템 (협업 구조의 시대로 폭발적인 업무 확장이 이루어짐)**

·Web 시스템의 혁명:

약 2000년대를 전후하여 Web 시스템으로 진화, 이는 서비스의 차원에서는 기업 내부의 업무를 이제는 기업 외에 오픈하는 혁신적인 단계이고, 이러한 기업 서비스의 대외 오픈은 C/S와 같은 자원 유지형(statefull)의 시스템적인 자원 한계를 갖기 때문에 한정된 자원을 공유하는 자원단절(stateless) 구조의 web 아키텍처로 발전하였으며, 기업 내부 사용자와는 다르게 외부 사용자(기업의 고객)의 환경이 가지고 있는 비표준의 환경에 적합한 메타(html)의 프로토콜을 지원하는 브라우저 기반의 클라이언트 환경으로 발전
즉 아래와 같은 환경적인 혁신의 단계
 - 인터넷의 탄생 : 내부 네트워크 --> 기업(국가 기관) 간의 네트워크 --> 상업적 기업 간의 네트워크 --> 대중의 네트워크 참여 --> 인터넷 환경
 - 기업 서비스의 외부로의 확장 : 기업의 확장, 인터넷을 통한 서비스의 시간적 공간적 제한의 극복 --> 폭발적 기업 서비스 증가
 - IT 자원의 대폭적 증가를 수용하는 아키텍처 : 제한적 자원의 할당 --> 서비스의 폭증 --> 불특정 다수 --> 클라이언트 표준화 불가능 / 트레픽 조절 불가능 --> 자원의 제한 --> 나누어 쓰는 자원의 공유 아키텍처 --> Web 아키텍처
외부 고객들에 시스템을 오픈하는 시스템

·Web의 한계 --> 가상화 기술의 등장:

 서비스의 지속적인 확장에 따른 Web으로 대표되는 IT아키텍처의 한계 발생, 기업의 IT 자원의 비용의 증가 발생
이 시점까지도 중앙의 서버를 중심으로하는 UNIX 아키텍처의 확장으로 환계 극복
  하지만 UNIX로 대표되는 인프라의 수직적 확장의 컴퓨팅 파워의 한계 및 비용의 한계에 도달 , 따라서 저비용 병렬 확장 구조의 아키텍처 필요성 발생 --> 가상화 기술을 통한 저비용 병렬 확장 아키텍처 도입
    가상화 기술은 하드웨어 가상화와 소프트웨어 가상화를 통해 물리적 리소스를 가상적으로 분리시키는 것을 가능하게 했습니다. 이는 물리적 서버 하나에서 여러 개의 가상 머신(Virtual Machines, VMs)을 실행할 수 있게 해주었습니다. 이것이 클라우드 컴퓨팅의 핵심 아이디어 중 하나입니다.
가상화를 통한 외부 대규모 고객들에 시스템을 오픈하는 시스템

·클라우드 컴퓨팅의 등장:

이러한 가상화 기술을 내부뿐만 아니라 외부에서 제공해 주는 서비스가 발생, 2000년대 후반, 클라우드 컴퓨팅 공급자(CSP)들이 가상화 및 자동화 기술을 기반으로 인프라스트럭처 및 서비스를 제공하기 시작했습니다. 이로써 기업과 개인은 필요에 따라 가상 서버 및 서비스를 더 쉽게 확장하고 사용할 수 있게 되었습니다.
인프라를 외부(내부) 서비스로 이용, 인프라 서비스 발생

·클라우드 서비스 모델의 다양화:

클라우드 컴퓨팅은 IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service) 등 다양한 서비스 모델을 제공하고, 사용자들은 필요한 서비스를 선택하여 이용할 수 있게 되었으며, 기업 내부에도 클라우드 서비스의 일부를 embedded할 수 있는 임대형 클라우드 서비스도 제공 , 해당 서비스는 보안 상으로 내부에 구축 해야 하는 경우, 그리고 일부 어플라이어스 솔루션 및 물리 장비를 통한 서비스의 경우, 매우 높은 레이턴시를 요구하는 서비스의 경우 내부에 구성이 필요 할 수 있다 . 임대형 클라우드는 AWS, Naver 등 CSP에서 제공하고 있음
인프라 서비스 모델의 다양화로 기업 서비스 환경의 full 클라우드화

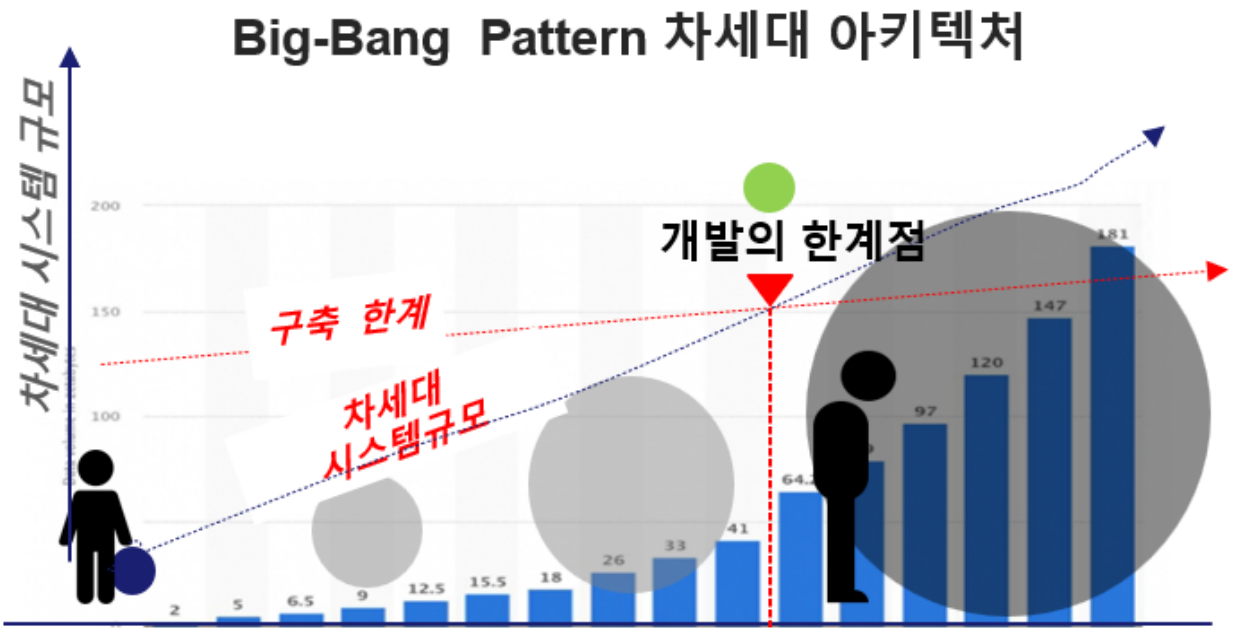
·하이브리드 및 멀티클라우드 전략:

기업의 모든 서비스가 시점 상 기간계까지 모두 클라우드 환경으로 전환할 수 없는 단계
각종 솔루션 및 보안 데이터, 그리고 DBMS, 특히 오라클에 대한 제한적 요건이 기업에서는 존재. 외부 클라우드의 구성 시에는 내부 직원 단말기 간의 지연(콜센터 음성 서비스 , 또는 교육 시스템의 동여상 전송, 많은 데이터의 download/upload)이 발생하는 경우는 하이브리드로 구성
  즉 On-Premise와 Off-Premise의 장점을 조합하고 다양한
클라우드 제공 업체의 장점을 활용하는 하이브리드 멀티 클라우드로 구성

클라우드 분류 및 특징

Big-Bang Pattern 차세대급 구축이 가능할까 ?

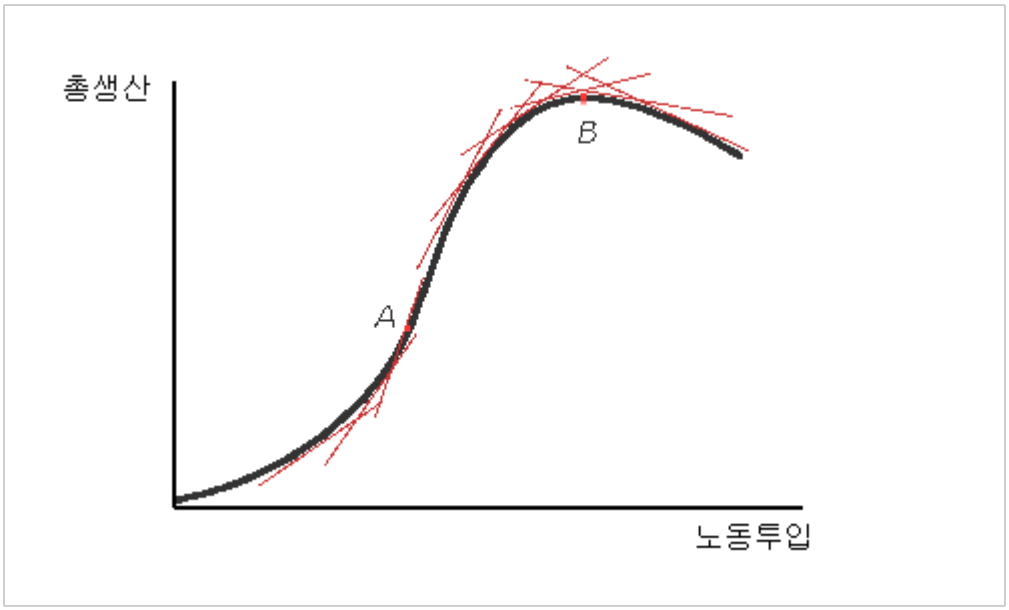
차세대를 클라우드로 해야 하는 이유



2010-2025년 전 세계 데이터 생성 및 캡처, 복제, 소비량 변화(단위: 제타바이트, 자료: 스태티스타 2021)

Big-Bang Pattern 차세대 구축의 한계

- 폭포수 방법론(waterfall approach) --> 차세대의 한계
- **한계 생산성의 체감** --> 일정 규모 이상 시스템 개발 한계성



- 규모 있는 개발 --> 상대적 높은 위험 요소 + 관리 요소 내포 --> 기하급수적 비용 증가

- 장기간 개발 후 다시 차세대 고려할 만큼 시장 변화 피동적

Separate of concern

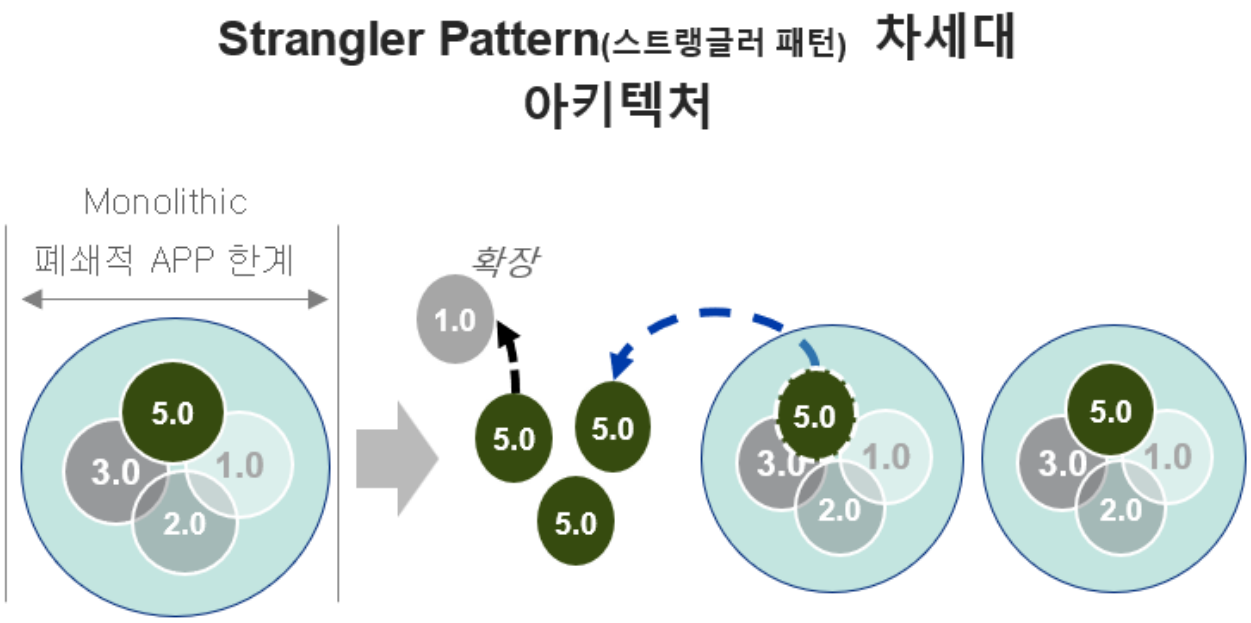
한 폴더에 모든 파일을 넣고 쓰시나요 ? 이것도 하나의 "관심사(관점)의 분리" 아닐까요 ?

번역을 "관점의 독립"이 어떨까

그럼 Legacy의 Modernization 이 필요해, 방법은 ?

[Separate of concern](#) [야금야금 작전](#) [Cohesion and Decoupling](#)

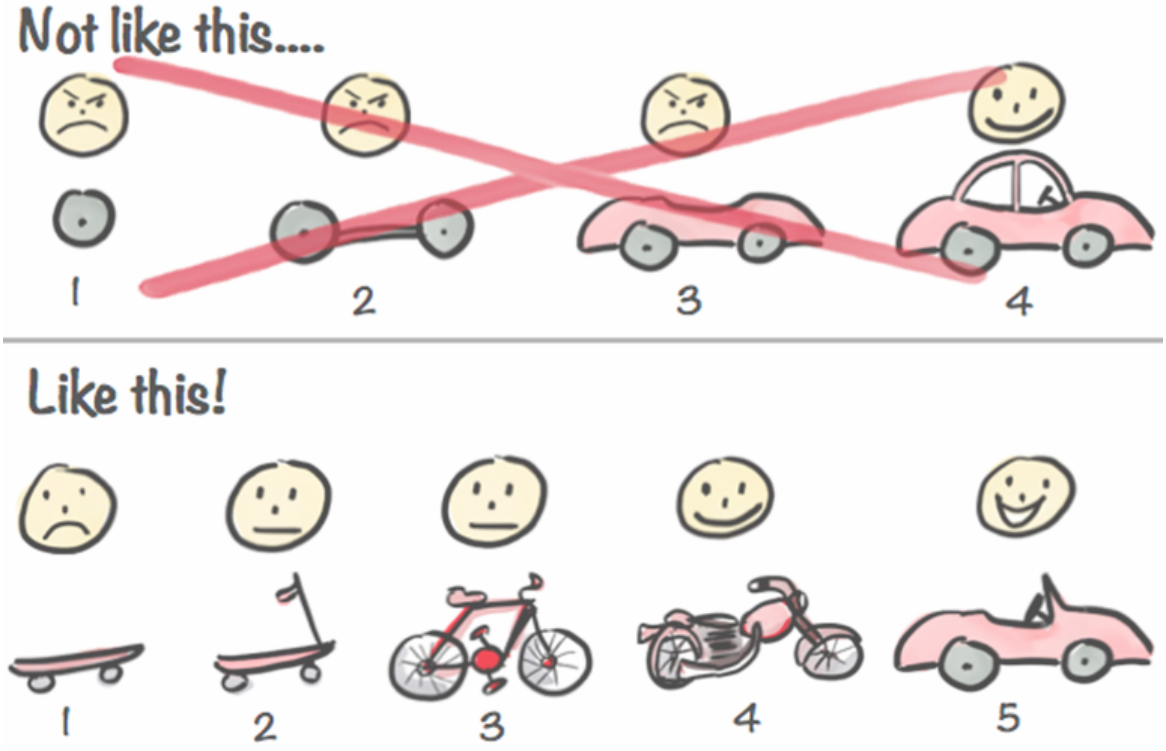
[Strangler Pattern(스트랭글러 패턴)](#)



Strangler Pattern 의 개발 과정 ,

- 전체가 아닌 부분의 적용
- (날렵,민첩이 아닌 , 측면)

• 성공에 기반한 확장 전략 ( )



• 지속적 플랫폼 도입

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

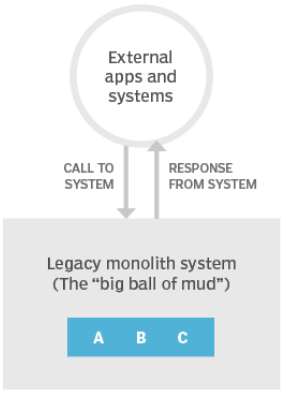
표면적으로 Strangler 패턴은 복잡해 보일 수 있습니다. 그러나 올바른 절차를 따른다면 실제로 구현하기가 매우 간단합니다.

아래 다이어그램은 Strangler 패턴 구현과 관련된 단계를 설명합니다.

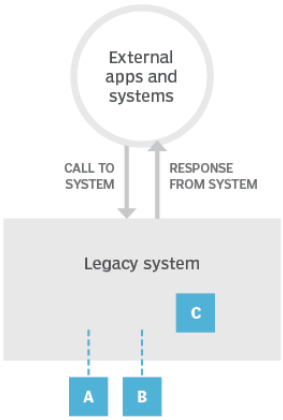
The Strangler Pattern

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

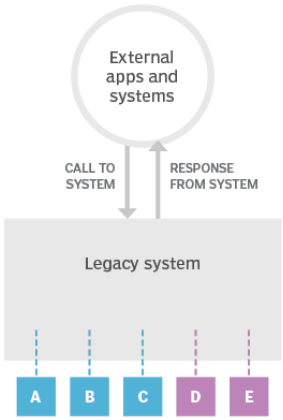
1. The system starts out as a tightly coupled "big ball of mud."



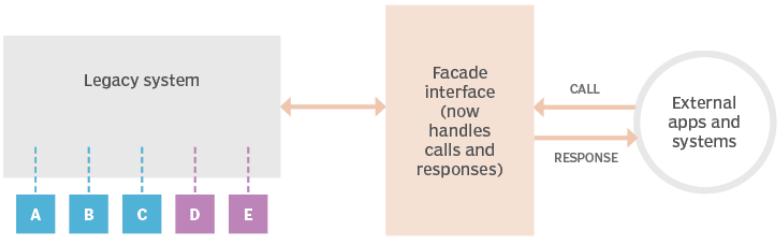
2. Begin to refactor code associated with specific service functions. Slowly turn blocks of refactored code into subsystem modules.



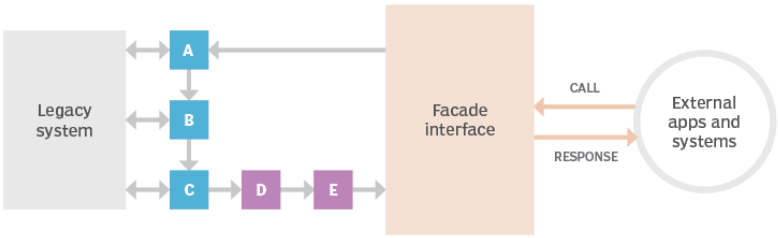
3. Add modernized features by creating new, testable subsystems that still align with existing subsystems.



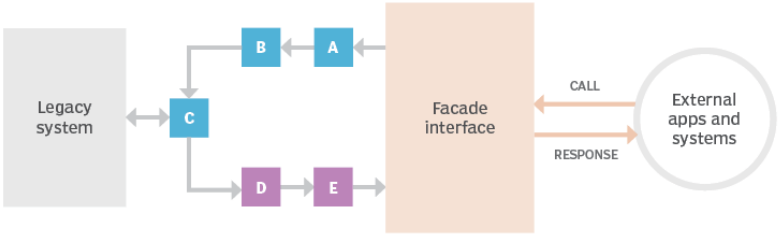
4. Create a facade interface that abstracts call and response processes from the legacy system. This interface now becomes the main point of contact for external apps and systems.



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