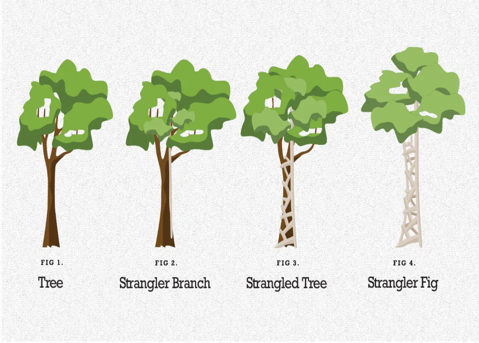
**Strangler Fig Pattern**

The Strangler Fig Application is based on an analogy to a vine that strangles a tree that it's wrapped around. The idea is that you use the structure of a web application-the fact that web apps are built out of individual URIs that map functionally to different aspects of a business domain

**Introduction to Strangler Fig Application**

The Strangler Fig Application is based on an analogy to a vine that strangles a tree that it's wrapped around. The idea is that you use the structure of a web application-the fact that web apps are built out of individual URIs that map functionally to different aspects of a business domain-to divide an application into different functional domains, and replace those domains with a new microservices-based implementation one domain at a time. This creates two separate applications that live side-by-side in the same URI space. Over time, the newly refactored application "strangles" or replaces the original application until finally you can shut off the monolithic application altogether. Thus, the Strangler Fig Application pattern steps are transform, coexist, and eliminate:

* **Transform**-Create a parallel new site (for example, in Openshift environment or even in your existing environment) but based on more modern approaches.
* **Coexist**-Leave the existing site where it is for a time. Redirect from the existing site to the new one so the functionality is implemented incrementally.
* **Eliminate**-Remove the old functionality from the existing site (or simply stop maintaining it) as traffic is redirected away from that portion of the old site.

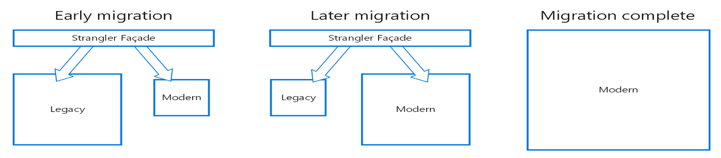


**Why We need Strangler Fig Pattern**

As systems age, the development tools, hosting technology (cloud migration), and even system architectures they were built on can become increasingly obsolete. As new features and functionality are added, the complexity of these applications can increase dramatically, making them harder to maintain or add new features to. Completely replacing a complex system can be a huge undertaking - A Big Bang Approach. Often, you will need a gradual migration to a new system, while keeping the old system to handle features that haven't been migrated yet. However, running two separate versions of an application means that clients have to know where particular features are located. Every time a feature or service is migrated, clients need to be updated to point to the new location.

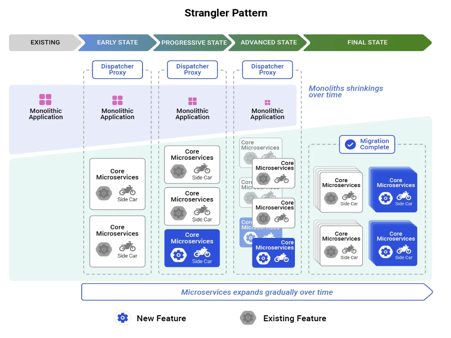
**How to Implement Strangler Pattern**

* Modernize an application by incrementally developing a new (strangler) application around the legacy application
* Incrementally replace specific pieces of functionality with new applications and services.
* Create a facade that intercepts requests going to the backend legacy system. The façade routes these requests either to the legacy application or the new services.
* Existing features can be migrated to the new system gradually, and consumers can continue using the same interface, unaware that any migration has taken place.
* Strangler pattern helps to minimize risk from the migration, and spread the development effort over time. With the façade safely routing users to the correct application, you can add functionality to the new system at whatever pace you like, while ensuring the legacy application continues to function.



**Benefits of Strangler Pattern**

* Provides a way to **reduce risk** when doing a system transformation.
* Keeps old services in play while refactoring to updated versions.
* Adds **uniquely new services** while refactoring older services.
* Allows you to push your changes in **small modular pieces**, easier for release
* Ensures **zero down time**
* Is generally more **agile**
* Easier **Rollbacks**



**Issues and Considerations**

* Consider how to handle services and data stores that are potentially used by both new and legacy systems. Make sure both can access these resources side-by-side.
* Structure new applications and services in a way that they can easily be intercepted and replaced in future strangler fig migrations.
* At some point, when the migration is complete, the strangler fig façade will either go away or evolve into an adaptor for legacy clients.
* Make sure the facade keeps up with the migration.
* Make sure the facade doesn't become a single point of failure or a performance bottleneck.

**When to use Strangler Fig Pattern**

* When gradually migrating a back-end application to a new architecture.
* When the legacy system is still in use and giving customer value.

**When NOT to use Strangler Fig Pattern**

* When requests to the back-end system cannot be intercepted.
* For smaller systems where the complexity of wholesale replacement is low.

**References**

* Martin Fowler Strangler Fig Application - <https://martinfowler.com/bliki/StranglerFigApplication.html>
* Confluent Strangler Fig - <https://developer.confluent.io/patterns/compositional-patterns/strangler-fig>