Agile with Elephants: Using the Strangler Pattern to Evolve and Consolidate Large Legacy Systems

Many IT organizations, particularly large ones, are familiar with the problem of "two-lane" IT – balancing the need to mitigate change and cost around legacy systems against the need to innovate at the speed of business. This problem rears its ugly head in particular when the maintenance or support costs of a legacy system outweigh the value, and when a merger or acquisition forces the consolidation of disparate but equally entrenched systems.

When these situations arise, IT leadership is challenged to bring monolithic systems into modernity, and often answers by attempting to re-write legacy systems wholesale – an undertaking fraught with business risk.

# The Elephant in the Room: Large-Scale Re-Writes aren't the Answer

There are several issues inherent in replacing legacy systems as on massive product

* **Almost Zero business value -- by design**. The most successful re-write is by nature going to produce a copy of the legacy system it seeks to replace, which at delivery time means it has added no value to the business. The existing concerns that prompted the re-write continue to exist for that much longer, because those concerns can't be addressed while maintaining parity with the existing system. Additionally, because the original development team may be gone and requirements may be scarce, this consistency will likely include the failings, and even the bugs, of the original system.
* **Large up-front investment.** Because re-writes have an all-in approach, the business is going to have to fully invest in that decision up-front, which includes planning, design, and staffing. If this sounds like the more costly and somewhat-antiquated waterfall method of development, that's because it is, with all the risks that entails.
* **Opportunity Cost.** For organizations that can't afford to staff for a re-write, undertaking a project of such magnitude means trade-offs against other value that those developers would be able to provide for the business. Re-writes often mean freezing the feature footprint of the legacy system, which is a built-in pain point as user concerns go unaddressed and key stakeholders don't see improvements. The business might also feel the pain of strategic opportunity cost due to locking itself into an execution of the re-write project despite business needs continually changing.
* **Re-Writes are prone to (public) failure.** Not only does the re-write approach increase the likelihood of a failure; the nature of the systems increases the impact that this risk can have when realized. Despite the costs involved in a re-write of a large system, even lackluster success isn't a sure thing, given the failure rate of large waterfall-style IT projects. Systems valued highly enough to warrant the costs of a re-write tend to be systems critical to the business – financial systems, ERP systems, and systems central to the business domain. When such keystone efforts fail, the visibility will likely extend to external clients or shareholders.

With these concerns, it's no wonder that in one of his most popular blog posts, Joel Spolsky[[1]](#footnote-1) lists complete re-writes under the title of "Things You Should Never Do".

# Eating the Elephant one Bite at a Time: the Strangler Pattern

Fortunately, IT organizations are not bound by these limiting approaches for evolving legacy systems. One popular approach to overcoming this challenge is the "Strangler Pattern". The term was coined by Martin Fowler, and originates from a type of vine that strangles its host tree over time until only a tree-shaped vine remains.

The steps to implement the strangler pattern are fairly straightforward:

* Build an interface or layer that exposes the legacy system
* Build a new piece of functionality that replaces part of the obsolete system
* Run both systems in parallel
* Repeat this process until the old system is entirely decommissioned.

Over time, because they duplicate the functionality but are more compelling to use, the new components and interfaces “strangle” the old legacy system, until the legacy system is reduced to a few small remaining components, and then can be completely decommissioned.

# Benefits of the Strangler Pattern

The Strangler approach yields a number of benefits.

* **A focus on value** **delivery**. The key benefit of the strangler pattern is that it enables value to be delivered during the re-write itself. New features can be developed alongside the legacy application, which means the business doesn’t have to sacrifice its priorities to achieve a re-write. In fact, a key tenet of a successful Strangler approach is to “make the [new] application so compelling that others will want to switch.”
* **Guaranteed ROI.** Because the Strangler pattern enables IT to deliver features to its users free of legacy constraints (and thus at an accelerated rate), the business will see a return on its investment even if the overall re-write effort is stalled or canceled. The all-or-nothing approach, and its associated drawbacks, can be traded for guaranteed value that expands over time.
* **Not locked into a specific approach**. Tackling a small piece of a legacy system at a time allows the business to respond to new and evolving information in an Agile fashion. If the initially selected approach fails for some reason, that failure is contained to a small piece and the approach can be improved upon.
* **Smaller feedback loops**. Rather than one large feedback loop on an entire re-write, the organization can take lessons learned from each of its smaller adaptations in order to improve the process over time. This will assist future phases of the Strangler effort and may offer insight into the development process at an organizational level. Additionally, real users are accessing new functionality at each turn, which may provide valuable insight into what users expect from the overall user experience and feature set of a system.
* **Risk reduction**. If a piece of the integration fails or is delayed, the business maintains the ability to utilize the legacy system. If the project is put on hold or abandoned, the organization does not have to live with the entire sunk cost of a large effort. Instead, it will have obtained incremental value without sacrificing access to the existing functionality it depends on

# References

* [Things You Should Never Do, Part I](http://www.joelonsoftware.com/articles/fog0000000069.html) [[Joel on Software](http://www.joelonsoftware.com/)]

1. Founder of the popular StackOverflow and Trello websites, as well as Fog Creek software. Joel is one of the longest-blogging programmers on the internet. [↑](#footnote-ref-1)