If It Ain't Broke, Don't Fix It

The name is usually an action phrase.

Intent: Save your reengineering effort for the parts of the system that will make a difference.

The intent should capture the essence of the pattern

Problem

The problem is phrased as a simple question. Sometimes the context is explicitly described.

Which parts of a legacy system should you reengineer?

This problem is difficult because:

- Legacy software systems can be large and complex.
- Rewriting everything is expensive and risky.

Yet, solving this problem is feasible because:

- Reengineering is always driven by some concrete goals.

Next we discuss the forces! They tell us why the problem is difficult and interesting. We also pinpoint the key to solving the problem.

Solution

The solution sometimes includes a recipe of steps to apply the pattern.

Only fix the parts that are "broken" — that can no longer be adapted to planned changes.

Tradeoffs

Each pattern entails some positive and negative tradeoffs.

Pros You don't waste your time fixing things that are not only your critical path.

Cons Delaying repairs that do not seem critical may cost you more in the long run.

Difficulties It can be hard to determine what is "broken".

There may follow a realistic example of applying the pattern.

Rationale

We explain why the solution makes sense.

There may well be parts of the legacy system that are ugly, but work well and do not pose any significant maintenance effort. If these components can be isolated and wrapped, it may never be necessary to replace them.

Known Uses

We list some well documented instances of the pattern.

Alan M. Davis discusses this in his book, *201 Principles of Software Development* .

Related Patterns

Related patterns may suggest alternative actions. Other patterns may suggest logical followup action.

Be sure to Fix Problems, Not Symptoms.

What Next

Consider starting with the Most Valuable First.