Agile Approach to a Legacy System

This paper summarizes the efforts of an XP team from ThoughtWorks Technologies to introduce new features to a legacy system. It provides a great informal guide to working with a legacy system using an agile development process (although not all the pointers necessitate an agile development process). This guide comes in the form of "rules of thumb" which are listed and briefly explained as follows:

- *Don't reproduce legacy code* - the first attempt to refactor the system was done by trying to rewrite the code in Java since it was spread out accross four different languages. Ultimately the rewrite approach failed (for reasons they explain in the paper) producing this rule.
- *Always ask the user what the problem is* - this allowed them to stay away from parts of the code that were irrelevant and gave them a clear focus for providing something valuable to the user
- *Refactor a legacy application by delivering new business value* - they describe their approach as being low risk because they "didn't change the legacy application and so the potential cost of failure was small. However the payoff for success was extremely high. The new system obsoletes legacy application functionality incrementally while delivering regular new features to users."
- *Incrementally build trust* - The idea here is to "prove that you can do the hardest part of the system." From there, each new feature added builds the trust of the customer.
- *Build a small, self-selected team* - Obviously this rule is not possible in many places for a variety of reasons so it was an advantage they were even able to have this option. Nevertheless they praise the fact that everyone on the team wanted to be there and that quality gave the team passion about their work.
- *Don't get hung up on process* - Their normal process was to have one iteration per week, but there were times when an iteration was needed before that. They made it happen without stalling their development.
- *Involve the whole team with larger refactorings so the team can move on as quickly as possible* - They describe their team discussions as "ego-less but opinionated." Everyone was willing to be wrong, but once a solution was decided on the ideas were owned by the group.
- *Effective teams need break points* - They describe how they would all take a break at a certain time each day, and go relax together.
- *Treat politics as a user requirement* - They built people's fears, anxieties, and doubts into their process by making them activities to be solved. They would all eventually be solved by the completion of other activities.
- *A System that connects to a legacy system must be tested using live feeds* - There was only so much that unit tests and simulations could provide them. They found many bugs that were in the system for months after they connected to live data.
- *Engage users and not only won't they turn it off, they will fight some of your battles for you* - They never told their users to stop using the legacy system, nor did they turn it off. They just focused on making the new features more compelling to use than the old system.
- *Don't waste a good team* - A good team is wasted when they don't have motivation and motivation, they discovered, comes from having new hard problems to solve.