EECS 394
SOFTWARE DEVELOPMENT

Chris Riesbeck

Cleaning Up
http://www.ambysoft.com/essays/deploymentTips.html
The MVP is running on developer machines. Now what?

- Feature freeze. Allocate a final iteration to
  - remove unused code
  - clean up existing code
  - upgrade components and libraries
  - re-test everything
  - write install scripts, if appropriate
  - work with their IT on the initial install
    - IT does everything, takes notes, asks questions

Monday, December 3, 2012
CLEANING UP

Technical Debt
"the continuous accumulation of shortcuts, hacks, duplication, and other sins we regularly commit against our code base in the name of speed and schedule"

The Agile Samurai, Chapter 13
"the continuous accumulation of shortcuts, hacks, duplication, and other sins we regularly commit against our code base in the name of speed and schedule"

The Agile Samurai, Chapter 13
Technical Debt Is More Than Just Code

While most of our technical debt is code centric, you can also have it in data and build and configuration files.

I was once part of a large-scale rewrite for a back-end system where a city name was spelled two different ways. The cost of this seemingly small difference was huge. Instead of not caring how the city was spelled, they had to write and carry this extra code and complexity for as long as that system remained in production, which for mainframe systems can be a very long time.
too much debt accumulation

Debt → Refactor

Weekly

Daily

Minutes

you want to be refactoring here
Refactoring

Rename
Extract method
Inline variable

"small incremental design improvements to your software without changing the overall external behavior"

Do only when all tests pass.

Martin Fowler's catalog:

The Agile Samurai, Chapter 13

Monday, December 3, 2012
SWYM  Say what you mean

Replace generic names with meaningful ones
Replace ad hoc names with conventional ones
Replace literal values with named constants
Replace calculations with well-named function calls

DRY  Don't repeat yourself

Replace repeated calculations with variables
Replace repeated calculation patterns with function calls
Remove redundant operations
SWYM  Say what you mean

Replace generic names with meaningful ones

double a;
int num[];

Replace ad hoc names with conventional ones

for (int counter = 0; ... 

Replace literal values with named constants

... / 12
... / 12
SWYM  Say what you mean

Replace generic names with meaningful ones

```plaintext
double a;          double average;
int num[];
```

Replace ad hoc names with conventional ones

```plaintext
for (int counter = 0; ...
```

Replace literal values with named constants

```plaintext
... / 12
... / 12
```
SWYM
Say what you mean

Replace generic names with meaningful ones

```c
double a;
int num[];
```

```c
double average;
int sums[];
```

Replace ad hoc names with conventional ones

```c
for (int counter = 0; ...
```

Replace literal values with named constants

```c
... / 12
```
SWYM  
Say what you mean

Replace generic names with meaningful ones

double a;  →  double average;
int num[];  →  int sums[];

Replace ad hoc names with conventional ones

for (int counter = 0; ...  →  for (int i = 0; ...;

Replace literal values with named constants

... / 12
... / 12
SWYM

Say what you mean

Replace generic names with meaningful ones

- `double a;` → `double average;`
- `int num[];` → `int sums[];`

Replace ad hoc names with conventional ones

- `for (int counter = 0; ...` → `for (int i = 0; ...`

Replace literal values with named constants

- `... / 12` → `... / eggsPerCarton`
- `... / 12`
SWYM  Say what you mean

Replace generic names with meaningful ones

double a;  \rightarrow  double average;
int num[];  \rightarrow  int sums[];

Replace ad hoc names with conventional ones

for (int counter = 0; ...  \rightarrow  for (int i = 0; ... 

Replace literal values with named constants

... / 12  \rightarrow  ... / eggsPerCarton
... / 12  \rightarrow  ... / monthsPerYear

Monday, December 3, 2012
SWYM
Say what you mean

Replace calculations with well-named function calls

```
interest = round(100 * balance * iy / 12) / 100;
```

Name functions with verb phrases that say what, not how

http://www.cs.northwestern.edu/academics/courses/325/readings/names.html
SWYM  Say what you mean

Replace calculations with well-named function calls

```plaintext
interest = round(100 * balance * iy / 12) / 100;

interest = roundToCents(balance * iy / 12);
```

Name functions with verb phrases that say what, not how

http://www.cs.northwestern.edu/academics/courses/325/readings/names.html
Don't repeat yourself

Replace repeated calculations with variables
Replace repeated calculation patterns with function calls

```c
if ( getRank(piles[MAX_PLAYERS][ci]) < getRank(piles[MAX_PLAYERS][i]) ) {
    if ( isLegalMatch(piles[player][0], piles[MAX_PLAYERS][i]) ) { ??? }
```
DRY
Don't repeat yourself

Replace repeated calculations with variables
Replace repeated calculation patterns with function calls

double xRange = xmax - xmin;
double yRange = ymax - ymin;
xmin -= ( xRange * ( factor - 1 ) / 2 );
xmax += ( xRange * ( factor - 1 ) / 2 );
ymin -= ( yRange * ( factor - 1 ) / 2 );
ymax += ( yRange * ( factor - 1 ) / 2 );
public bool DealerWins(Hand hand1) {
    var h1 = hand1; int sum1 = 0;
    foreach (var c in h1) {
        sum1 += Value(c.Value, h1);
    }
    var h2 = DealerManager.Hand; int sum2 = 0;
    foreach (var c in h2) {
        sum2 += Value(c.Value, h2);
    }
    if (sum2 >= sum1) {
        return true;
    }
    else
        return false;
}
return false;
Ooops! We weren't refactoring. Should we go back and clean up everything? Is that a user story?
Normally, avoid stories and iterations that are only refactoring.
Technical Debt

Normally, avoid stories and iterations that are only refactoring.

Instead, add unit tests and refactor only the code that you need to modify to add user functionality.
Normally, avoid stories and iterations that are only refactoring.

Instead, add unit tests and refactor only the code that you need to modify to add user functionality.

This amortizes the cost and avoids needless cleanup.

The Agile Samurai
Chapter 13
pp. 223-225