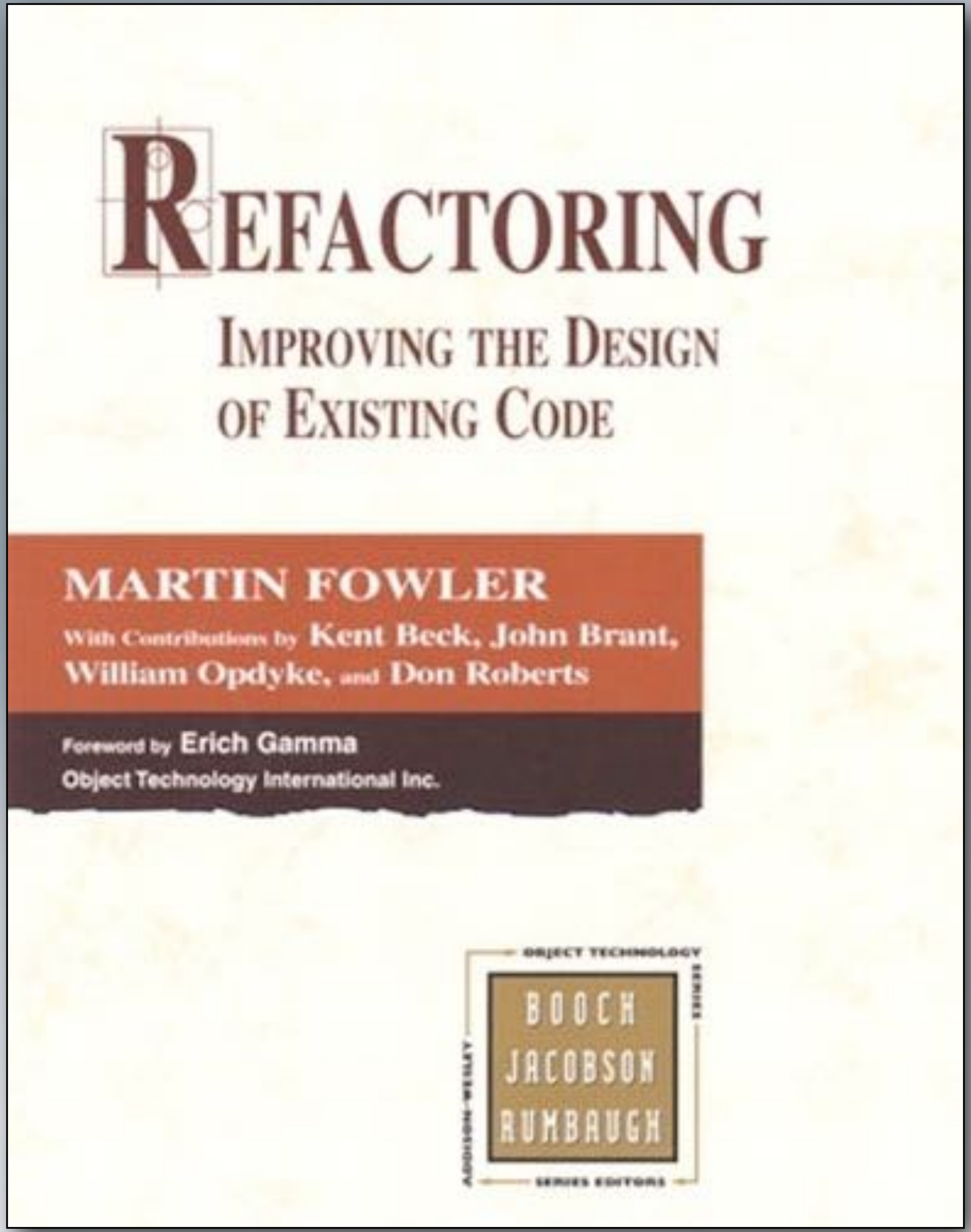
It Was Like That When I Got Here:

Steps Toward Modernizing A Legacy Codebase

@pmjones

mlaphp.com

Read These



REFACTORING

IMPROVING THE DESIGN OF EXISTING CODE

MARTIN FOWLER

With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts

Foreword by Erich Gamma

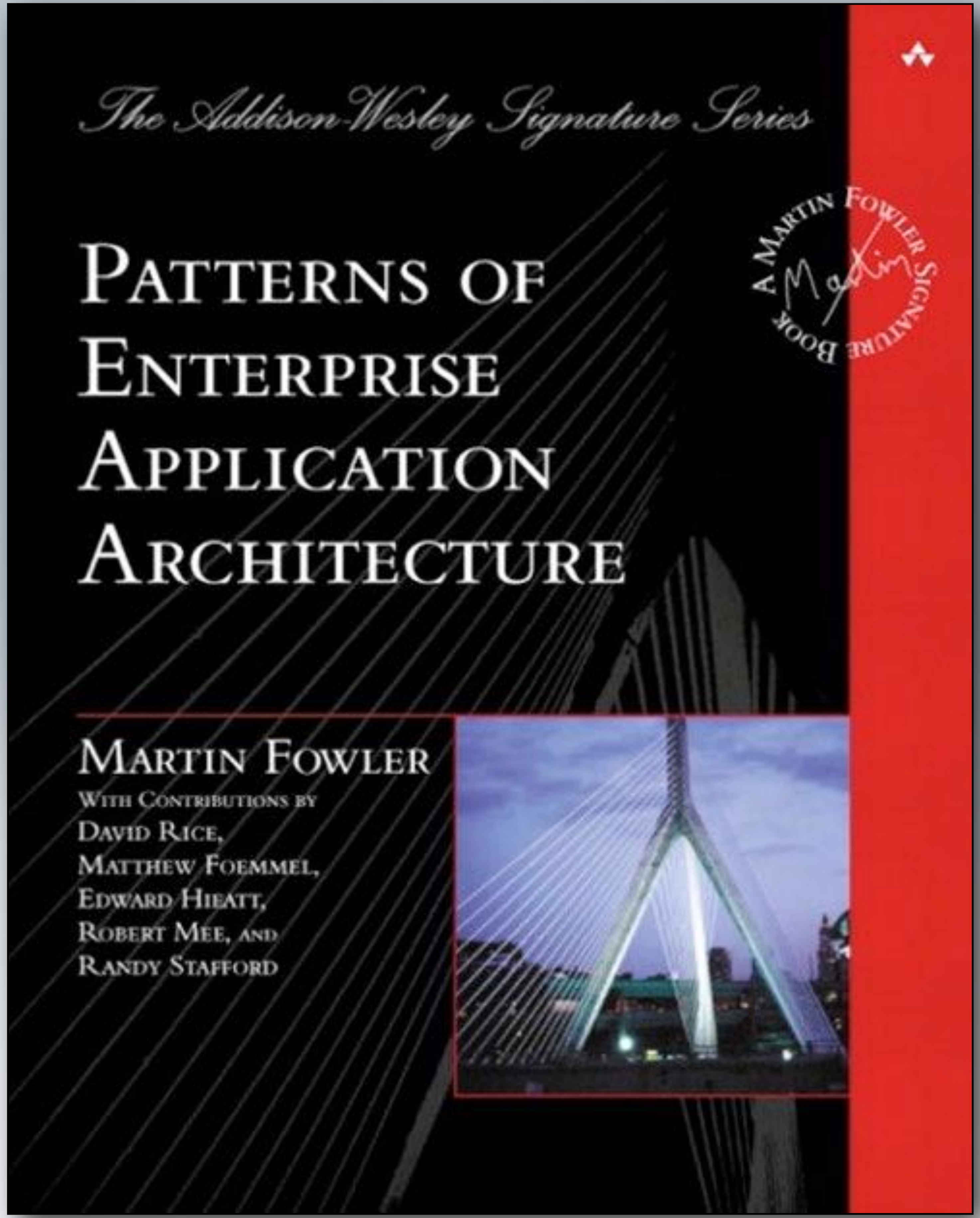
Object Technology International Inc.

OBJECT TECHNOLOGY SERIES

BOOCH JACOBSON RUMBAUGH

ADDISON-WESLEY

SERIES EDITORS



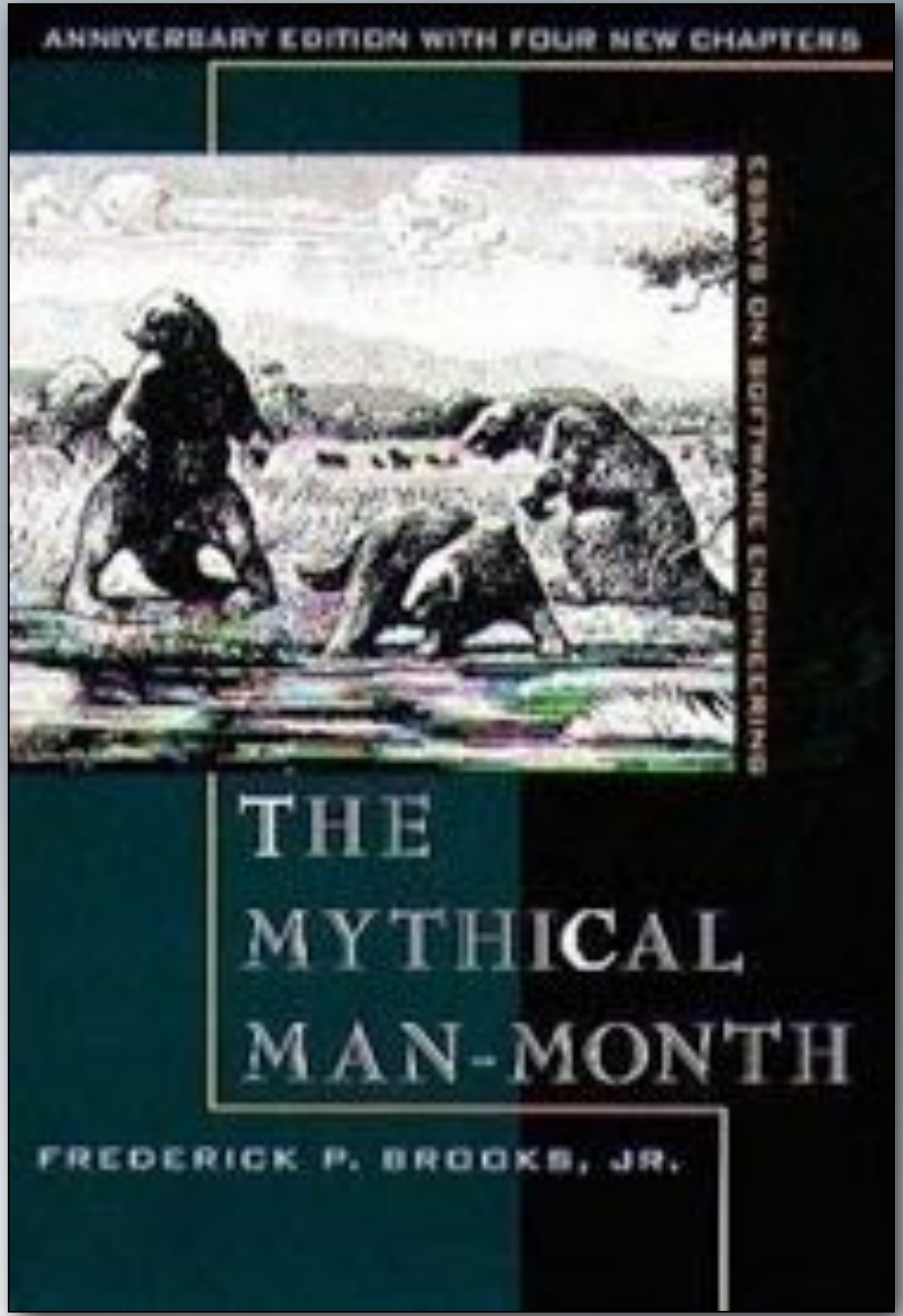
The Addison-Wesley Signature Series

PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE

A MARTIN FOWLER SIGNATURE BOOK

MARTIN FOWLER

WITH CONTRIBUTIONS BY DAVID RICE, MATTHEW FOEMMEL, EDWARD HIEATT, ROBERT MEE, AND RANDY STAFFORD



ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS

ESSAYS ON SOFTWARE ENGINEERING

THE MYTHICAL MAN-MONTH

FREDERICK P. BROOKS, JR.

About Me

- 8 years USAF Intelligence
- BASIC in 1983, PHP since 1999
- Jr. Developer, VP Engineering
- Aura project, Zend_DB, Zend_View
- ZCE Advisory Board
- PHP-FIG: PSR-1, PSR-2, PSR-4
- Action-Domain-Responder

Overview

- The code you are suffering with
- Incremental reductions of technical debt
- Life is better but still room for improvement

It Was Like That When I Got Here

Messy Codebase

- Page scripts in docroot (page-based)
- Spaghetti include logic (include-oriented)
- Few or no classes
- Global variables
- No unit tests -- QA working overtime

No Time To Remedy

- Bugs to fix, *right now*
- Features to implement, *right now*
- Making your own life easier?
  Not a priority.
- Dig in and try to make do
- How did it get this bad?
  “It was like that when I got here.”

The Great Thing About PHP ...

- ... is that anyone can use it.
- Have an idea? Implement it!
- It works! Great success!
- ... it "works."

The Awful Thing About PHP ...

- ... is that anyone can use it.
- The codebase is like a “dancing bear”
- Architecture? Maintenance? Testing?
- Move on to the next idea ...
- ... but **you** are stuck with it now.

Typical Page Script

see editor for example

Why Is It Like This?

- Original developer probably didn’t know better
- Subsequent developers worked with what was there
- “We can fix it later ...”
- ... until later becomes now.

Technical Debt

- A metaphor referring to the eventual consequences of poor or evolving software architecture and software development within a codebase.
- As a change is started on a codebase, there is often the need to make other coordinated changes at the same time in other parts of the codebase.
- http://en.wikipedia.org/wiki/Technical_debt

Paying Off Technical Debt

Paying Off Technical Debt

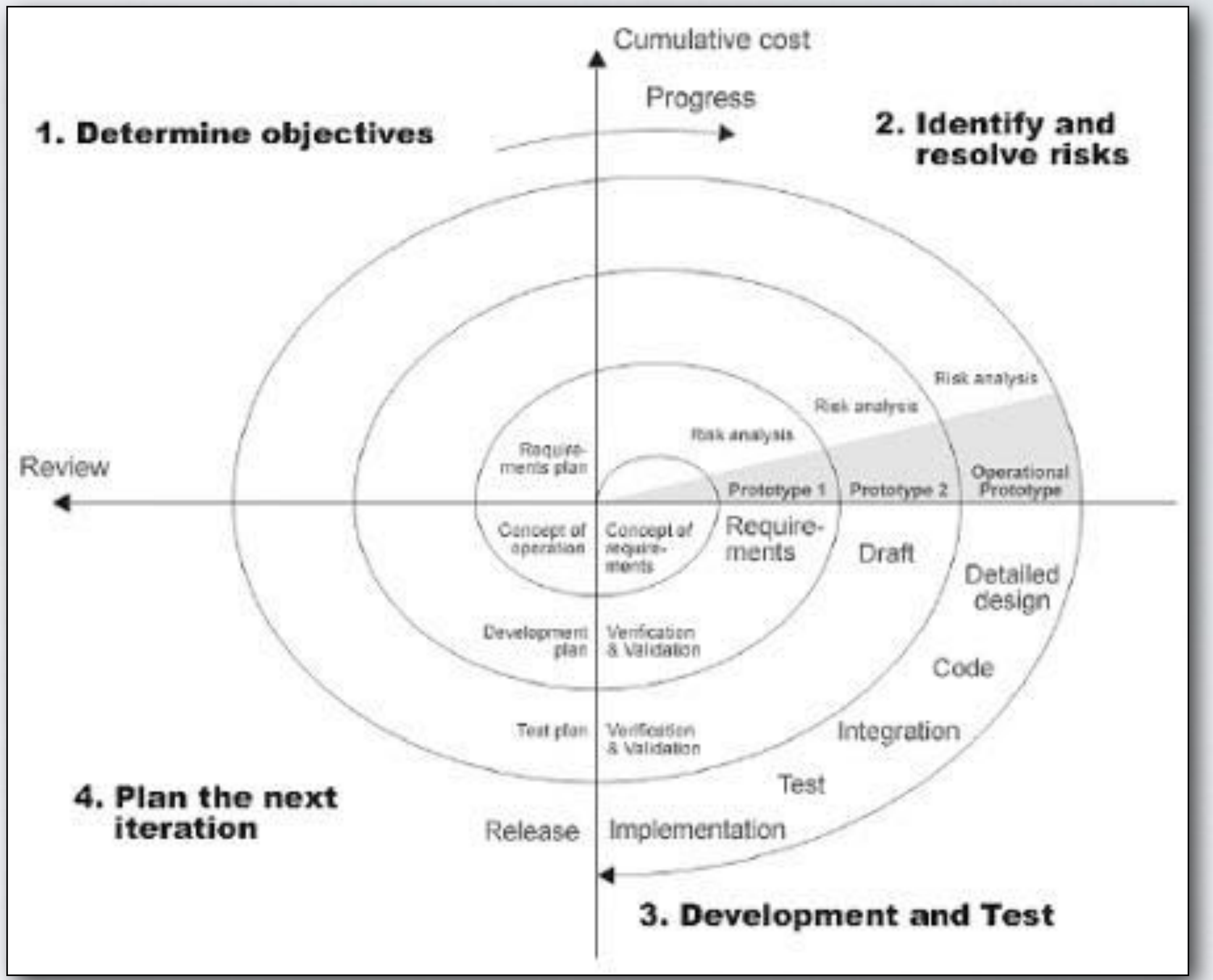
- A lot like paying off financial debt
- Got the stuff first, but have to pay for it eventually
- Must pay off technical debt not of our own choosing
- Suffer as things are, or suffer through change

Declare Bankruptcy

- Rewrite from scratch!
- Expend effort while not earning revenue
- Old devs on new project?
  New devs on new project?
- Takes longer than you think
- End up with different bad architecture

Incremental Approach

- Pay off smallest debt first (build inertia and raise spirits)
- Small changes across codebase
- Build on previous small changes
- Improve quality over time



Incremental Goals

- Keep the application running
- Consolidate classes for autoloading (PSR-0)
- Convert globals to injected dependencies
- After each change: “spot check”, commit, push, QA

Consolidate Classes For Autoloading

What Is Autoloading?

```
// without autoloading, must include file first
include_once "/path/to/classes/Example/Name.php";
$obj = new Example_Name();

// with autoloading, gets included automatically
$obj = new Example_Name();
```

PSR-0

- Class name maps directly to file name
- Namespace separators map to directory separators
- Class underscores map to directory separators
- `Vendor\Package_Name\Example_Name`
  `=> Vendor/Package_Name/Example/Name.php`

```php
function autoload($class)
{
    $class = ltrim($class, '\\');
    $file  = '';
    $ns    = '';
    $pos = strripos($class, '\\')
    if ($pos) {
        $ns    = substr($class, 0, $pos);
        $class = substr($class, $pos + 1);
        $file  = str_replace('\\', DIRECTORY_SEPARATOR, $ns)
               . DIRECTORY_SEPARATOR;
    }
    $file .= str_replace('_', DIRECTORY_SEPARATOR, $class);

    $base = "/path/to/classes";
    require "{$base}/{$file}.php";
}

spl_autoload_register('autoload');
```

Move Class Files

- If you have class files in several paths, move to same base path
- If you have more than one class per file, split into separate files
- If you define classes as part of a script, extract to own file
- Remove `include/require` as you go (grep)
- If needed, change names as you go (grep)

Convert Function Files To Class Files

- Many projects have files of function definitions
- Wrap in a class as static or instance methods
- Move to classes directory
- Change calls to static or instance calls (grep)
- Remove `include/require` as you go (grep)

Original Function

```php
function fetch_results()
{
    global $db;
    $results = $db->fetch('whatever');
    return $results;
}

$results = fetch_results();
```

Static Method

```
class Example
{
    public static function fetchResults()
    {
        global $db;
        $results = $db->fetch('whatever');
        return $results;
    }
}

$results = Example::fetchResults();
```

Instance Method

```
class Example
{
    public function fetchResults()
    {
        global $db;
        $results = $db->fetch('whatever');
        return $results;
    }
}

$example = new Example;
$results = $example->fetchResults();
```

Convert Globals to Injected Dependencies

Instantiating Dependencies In Methods

```
class Example
{
    public function fetchResults()
    {
        $db = new Database('username', 'password');
        return $db->fetch('whatever');
    }
}
```

Drawbacks Of Method Instantiation

- New connection on each call
- Cannot reuse connection
- Parameter modification

Global Dependencies

```
// setup file
$db = new Database('username', 'password');

// example class file
class Example
{
    public function fetchResults()
    {
        global $db;
        return $db->fetch('whatever');
    }
}
```

Global Drawbacks

```php
class Evil
{
    public function actionAtADistance()
    {
        global $db;
        unset($db);
    }
}
```

Dependency Injection

- Instead of reaching out from inside the class to bring in dependencies ...
- ... inject the dependency into the class from the outside.

Starting Point: Global In Method

```
class Example
{
    public function fetchResults()
    {
        global $db;
        return $db->fetch('results');
    }
}
```

Interim: Global In Constructor

```
class Example
{
    public function __construct()
    {
        global $db;
        $this->db = $db;
    }

    public function fetchResults()
    {
        return $this->db->fetch('results');
    }
}
```

Final: Dependency Injection

```php
class Example
{
    public function __construct($db)
    {
        $this->db = $db;
    }

    public function fetchResults()
    {
        return $this->db->fetch('results');
    }
}
```

Change Instantiation Calls

- Must change all `new` instantiations to pass dependencies (grep)
- Class instantiation inside methods? Pass intermediary dependencies.

Intermediary Dependency

```
class Example
{
    public function fetchResults()
    {
        global $db;
        return $db->fetch('whatever');
    }
}

class Service
{
    public function action()
    {
        $example = new Example;
        return $example->fetchResults();
    }
}
```

```php
class Example
{
    public function __construct($db)
    {
        $this->db = $db;
    }
    public function fetchResults()
    {
        return $this->db->fetch('whatever');
    }
}

class Service
{
    public function __construct($db)
    {
        $this->db = $db;
    }
    public function action()
    {
        $example = new Example($this->db);
        return $example->fetchResults();
    }
}
```

Eliminate Intermediary Dependency

```php
class Service
{
    public function __construct($example)
    {
        $this->example = $example;
    }

    public function action()
    {
        return $this->example->fetchResults();
    }
}
```

Progression of Instantiation

```php
// all globals
$service = new Service;

// intermediary: Example uses DI,
// but Service creates Example internally
$db = new Database('username', 'password');
$service = new Service($db);

// all DI all the time
$db = new Database('username', 'password');
$example = new Example($db);
$service = new Service($example);
```

Life After Reorganizing

Initial Goals Completed ...

- Consolidated into classes with PSR-0 and autoloading
- Removed globals in favor of dependency injection
- Kept it running the whole time
- Paid off some technical debt
- Organizational structure for future work
- Start writing unit tests

WE ALL TEST DOWN

(with apologies to Stephen King's "It")

... But Much Remains

- Using `new` keyword
- Embedded SQL statements
- Embedded domain logic
- Embedded presentation logic
- Embedded action logic
- Embedded include calls
- Router + front controller
- DI container

leanpub.com/mlaphp

Autoloaded,
Dependency Injected,
Unit Tested,
Layer Separated,
Front Controlled