Brakeman and Jenkins: The Duo Detects Defects in Ruby on Rails Code

Justin Collins
Tin Zaw

AppSec USA
September 23, 2011

About Us

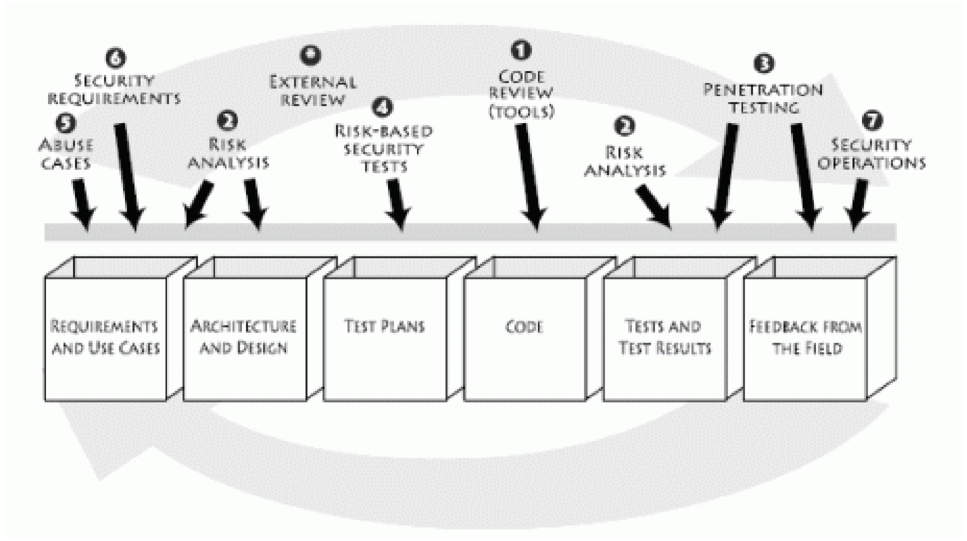
Justin Collins - @presidentbeef

Tin Zaw - @tzaw

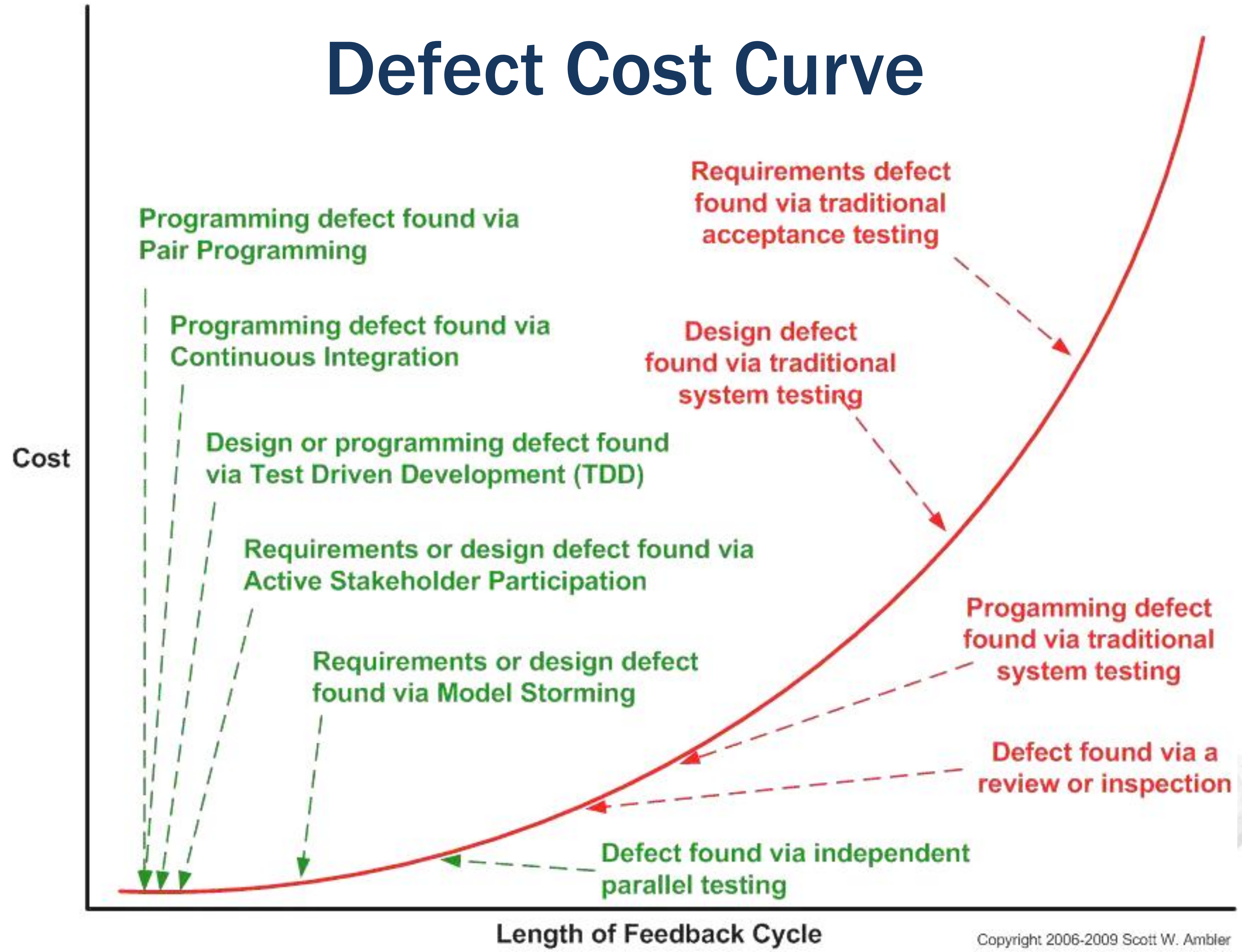
Our Philosophy: Light Touch

Use *tools* to detect and report
security defects in code
early in the development cycle
with *minimal impact*
to development workflow

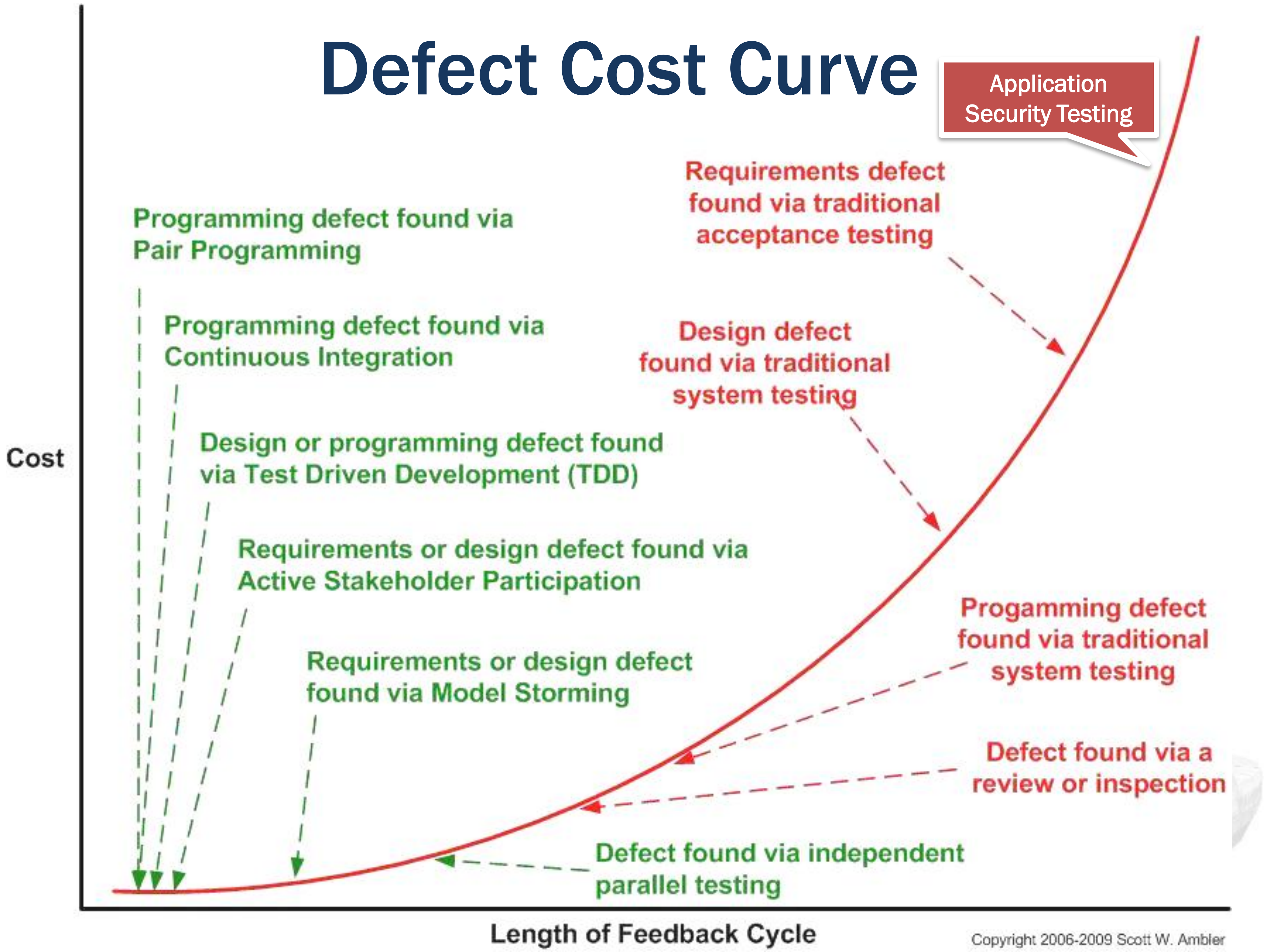
McGraw's Touch Point #1 Code Review (Tools)



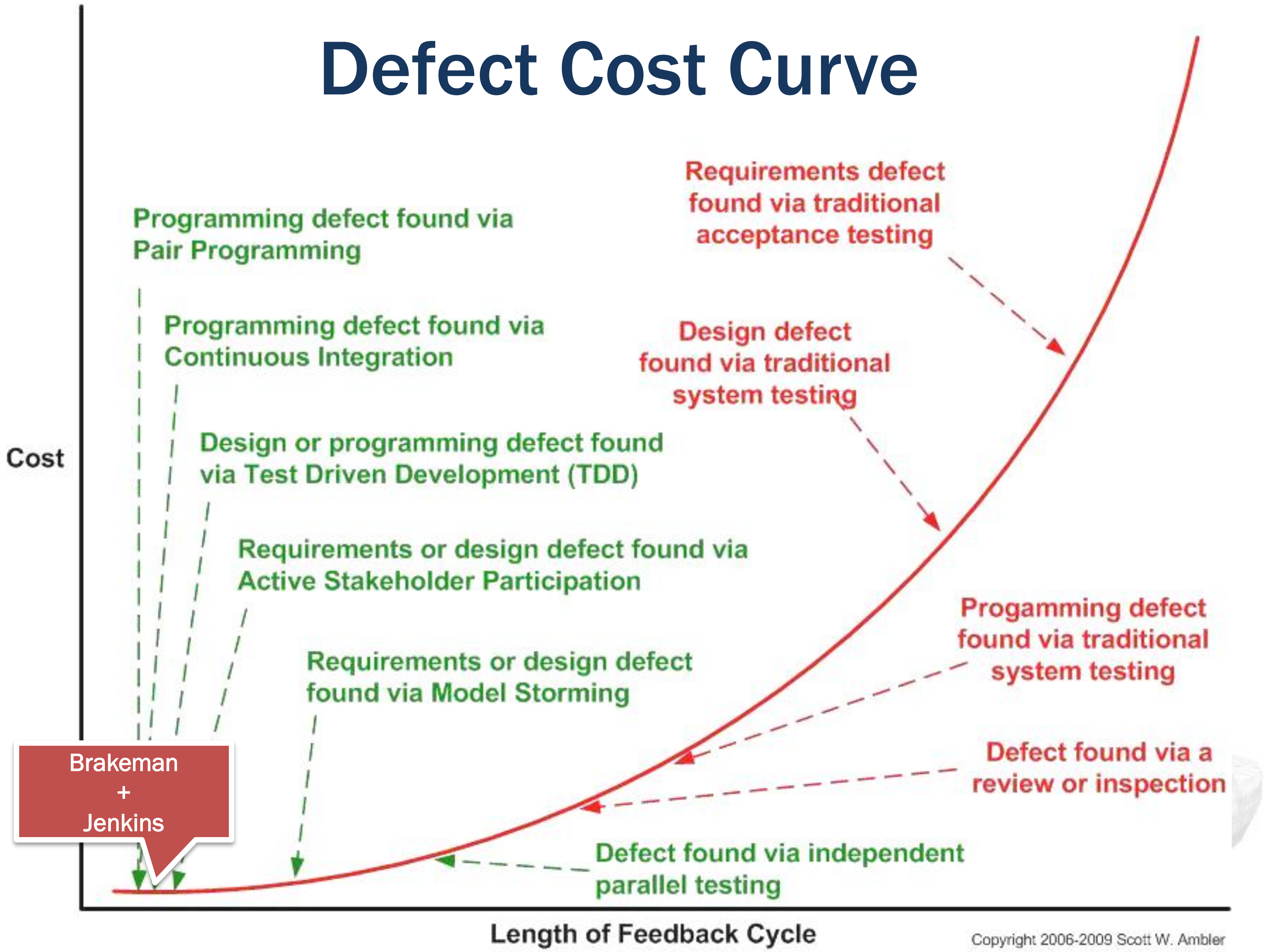
Defect Cost Curve



Defect Cost Curve



Defect Cost Curve


Cost
Length of Feedback Cycle
Programming defect found via Pair Programming
Programming defect found via Continuous Integration
Design or programming defect found via Test Driven Development (TDD)
Requirements or design defect found via Active Stakeholder Participation
Requirements or design defect found via Model Storming
Defect found via independent parallel testing
Defect found via a review or inspection
Progamming defect found via traditional system testing
Design defect found via traditional system testing
Requirements defect found via traditional acceptance testing
Brakeman + Jenkins
Copyright 2006-2009 Scott W. Ambler

Static vs. Dynamic Analysis

- Penetration Testing Pros
  - Replicates real life deployment
  - Entire application stack, configuration

- Penetration Testing Cons
  - Reports symptoms, not root causes
  - Setup time, find defects late during QA cycle
  - Incomplete view of running app

Static vs. Dynamic Analysis

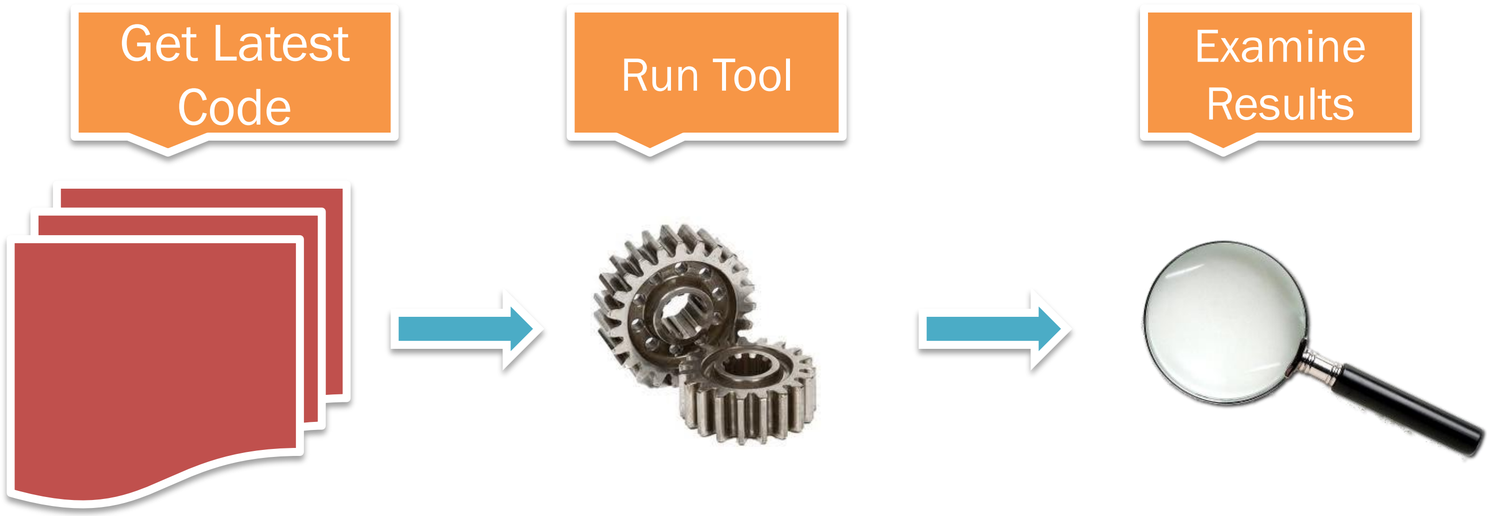
- Static Code Analysis Pros
  - Early detection of defects
  - Integrated into developer’s workflow
  - No deployment required

- Static Code Analysis Cons
  - Limited to code
  - Need access to source code

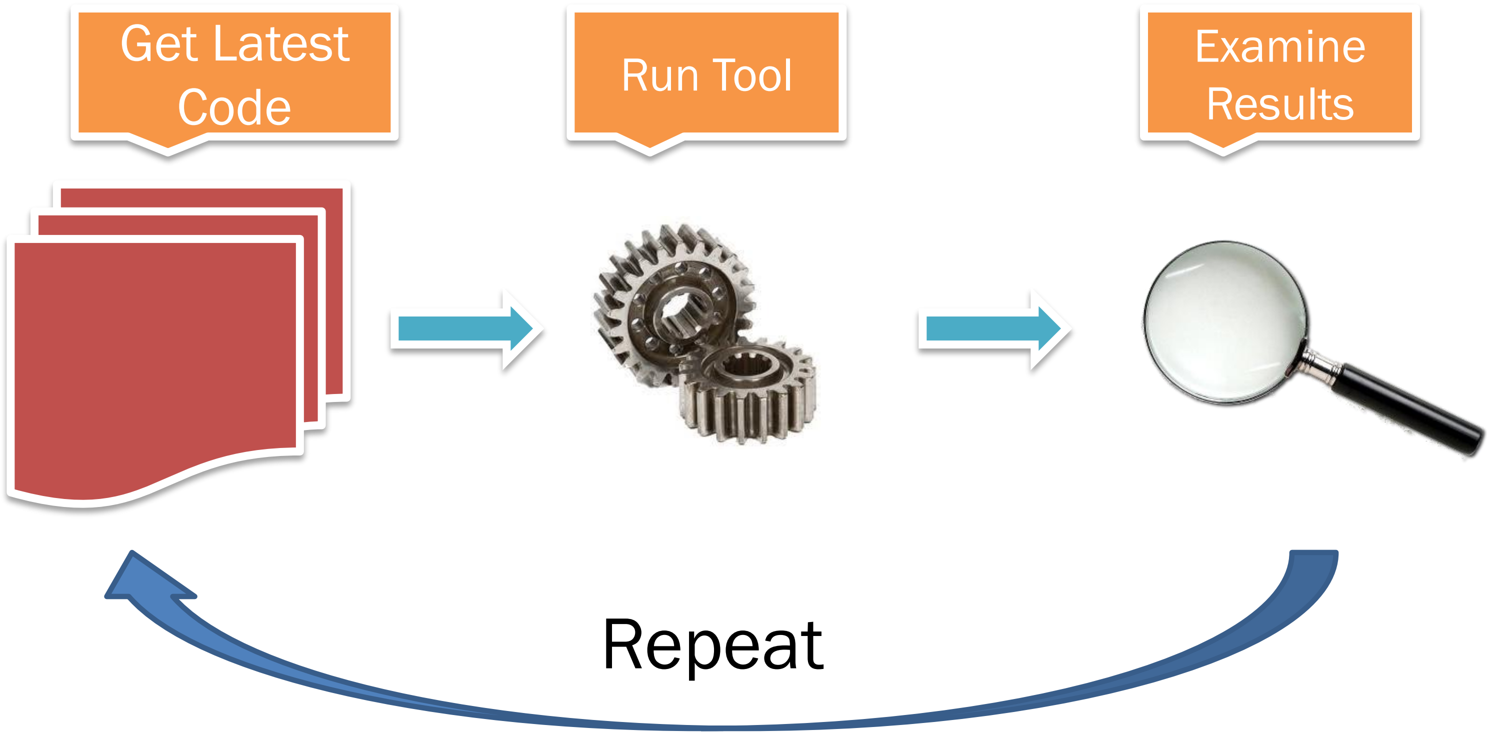
Existing Static Analysis Tools for Security Defects

C/C++	<many>
C#/.Net	<many>
Java	<many>
Ruby	?
Ruby on Rails	Brakeman

Manual Workflow



Manual Workflow



Automated Workflow

Brakeman

http://brakemanscanner.org

Ruby on Rails

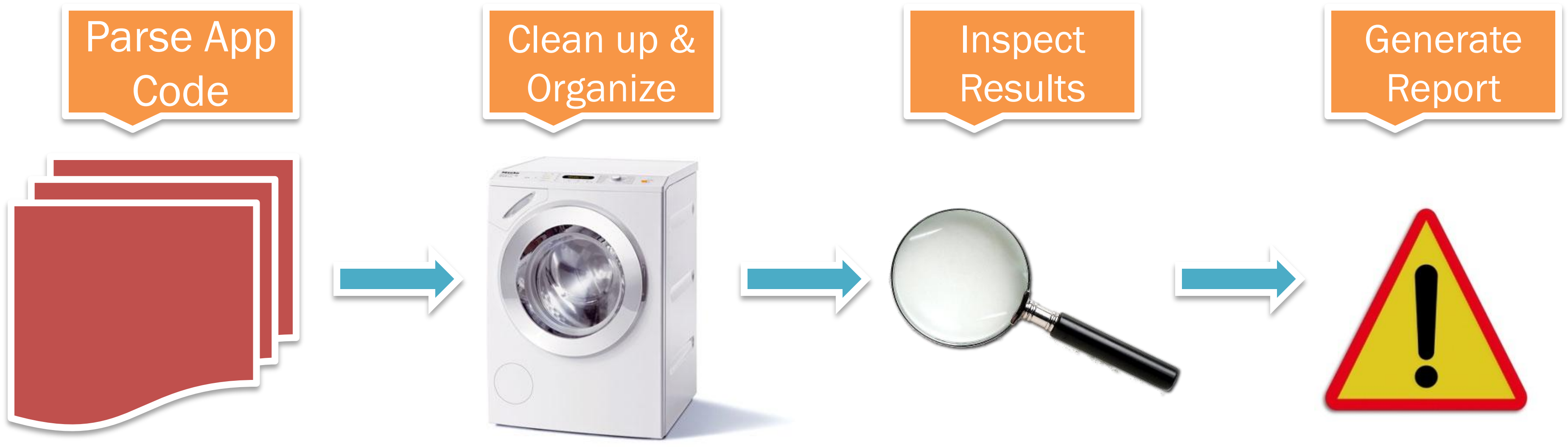
http://rubyonrails.org/

Web application framework using the Ruby language

Built on the model-view-controller design pattern

"Convention over configuration" – encourages assumptions which lead to default behavior

Brakeman Application Flow



Vulnerabilities Brakeman Detects

Cross site scripting

SQL injection

Command injection

Unprotected redirects

Unsafe file access

Default routes

Insufficient model validation

Version-specific security issues

Unrestricted mass assignment

Dangerous use of eval()

...and more!

Example: Cross Site Scripting (Rails 2.x)

```
<b>Results for <%= params[:query] %></b>
```

Example: Cross Site Scripting (Rails 3.x)

```
<b>Results for <%= raw params[:query] %></b>
```

Example: Cross Site Scripting (Rails 3.x)

```
<b>Results for <%= raw params[:query] %></b>
```

Unescaped parameter value near line 1: params[:query]

Example: SQL Injection

```
username = params[:user][:name]

User.find(:all,
  :conditions => "name like '%#{username}%'")
```

Example: SQL Injection

```
username = params[:user][:name]

User.find(:all,
   :conditions => "name like '%#{username}%'")
```

Possible SQL injection near line 87:
User.find(:all, :conditions => ("name like '%#{params[:user][:name]}%'")

Extended Example - Filters

```
class ApplicationController < ActionController::Base

  def set_user
    @user = User.find(params[:user_id])
  end

end
```

Method in application controller sets the @user variable

Extended Example - Filters

```ruby
class UserController < ApplicationController
  before_filter :set_user

  def show
  end

end
```

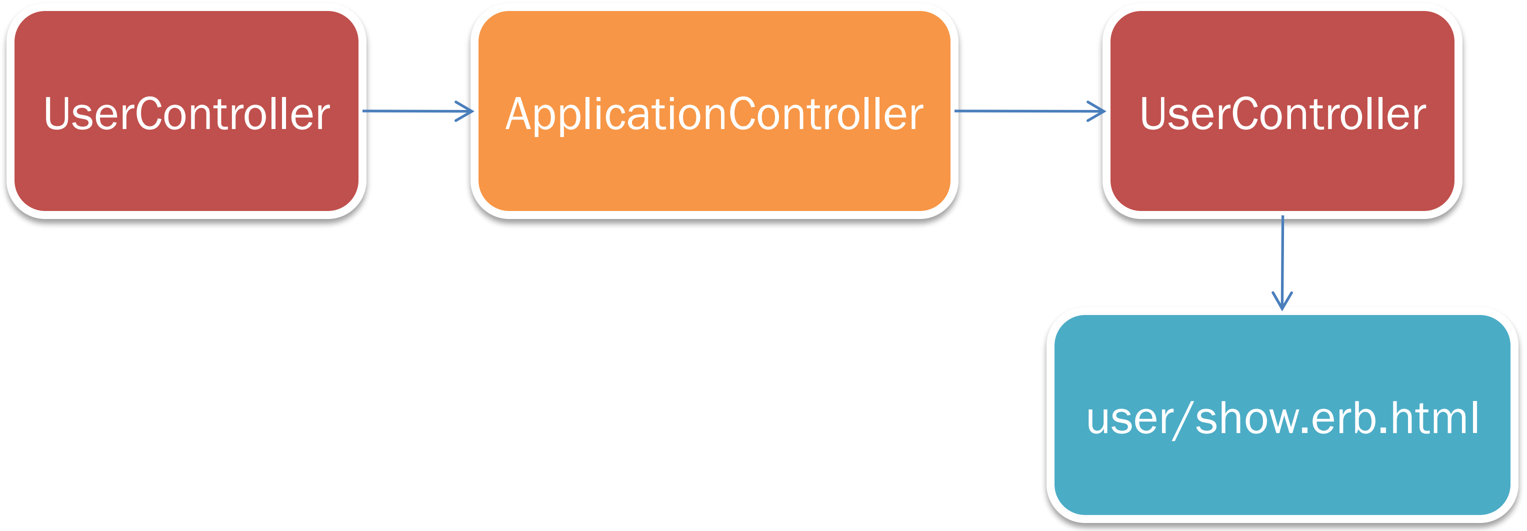
User controller calls set_user before any action

Extended Example - Filters

```
<%= raw @user.bio %>
```

View outputs the result of a method call on the @user variable

Extended Example - Filters



Data flow followed from filter through to the view

Extended Example - Filters

```
<%= raw @user.bio %>
```

Unescaped model attribute near line 5:
User.find(params[:id]).bio

Example: Mass Assignment

```
class User < ActiveRecord::Base
end
```

User model generated by Rails

Example: Mass Assignment

```ruby
class UsersController < ApplicationController
 #...
  def new
    @user = User.new(params[:user])
    #...
  end
end
```

Excerpt of Users controller generated by Rails

Example: Mass Assignment

```
class UsersController < ApplicationController
 #...
  def new
    @user = User.new(params[:user])
    #...
  end
end
```

Unprotected mass assignment near line 43:
User.new(params[:user])

Jenkins

Open source continuous integration server

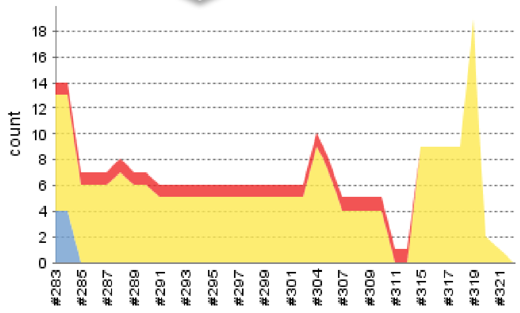
http://jenkins-ci.org

How Jenkins Works

Monitor Conditions

Run Jobs

Aggregate Results



How Jenkins Works

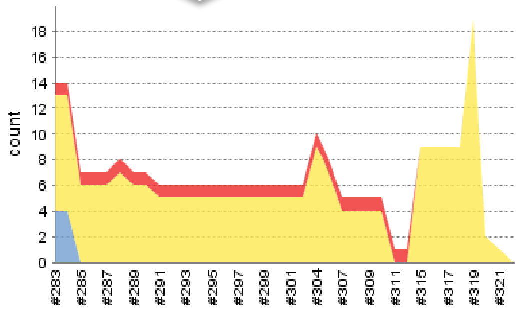
Monitor Conditions

```
git push
svn commit
```

Run Jobs

```
brakeman
```

Aggregate Results



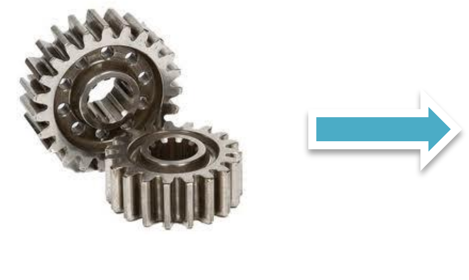
```
Security
Warnings
```

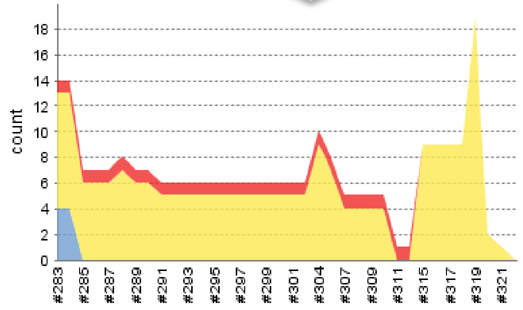
Brakeman Plugin for Jenkins

Run Brakeman

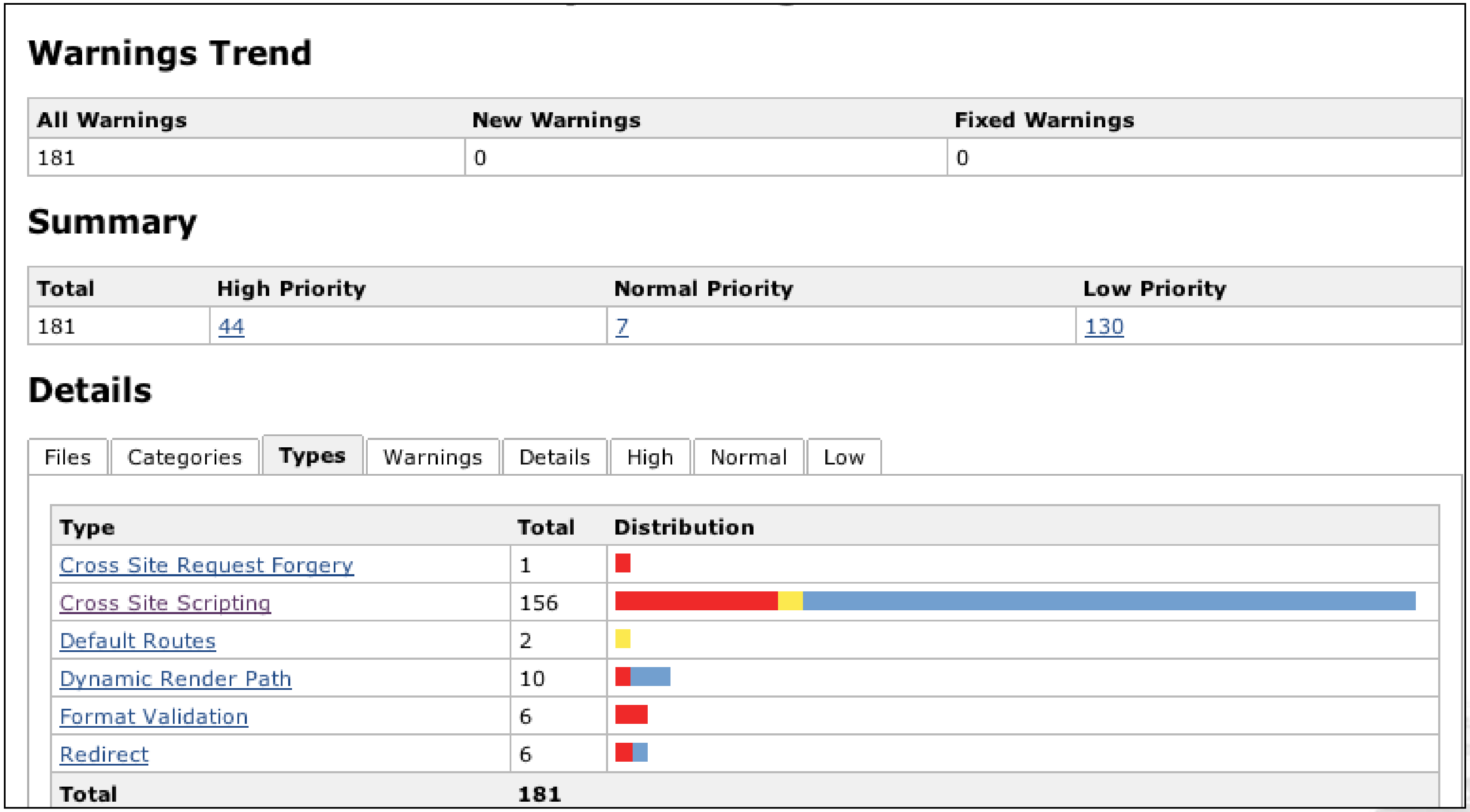
Collect Warnings

Generate Reports





Some Results



Warnings Trend

All Warnings	New Warnings	Fixed Warnings
181	0	0

Summary

Total	High Priority	Normal Priority	Low Priority
181	44	7	130

Details

Files | Categories | **Types** | Warnings | Details | High | Normal | Low

Type	Total	Distribution
Cross Site Request Forgery	1	🟥
Cross Site Scripting	156	🟥🟨🟦
Default Routes	2	🟨
Dynamic Render Path	10	🟥🟦
Format Validation	6	🟥
Redirect	6	🟥🟦
Total	**181**	

Using Brakeman

```
gem install brakeman
cd your/rails/app
brakeman
```

Resources

- Ruby
  - http://ruby-lang.org
- Ruby on Rails
  - http://rubyonrails.org
- Ruby on Rails Security Guide
  - http://guides.rubyonrails.org/security.html
- Brakeman
  - http://brakemanscanner.org
- Jenkins
  - http://jenkins-ci.org
- Brakeman plugin for Jenkins
  - http://github.com/presidentbeef/brakeman-jenkins-plugin