Enforcing Jenkins Best Practices

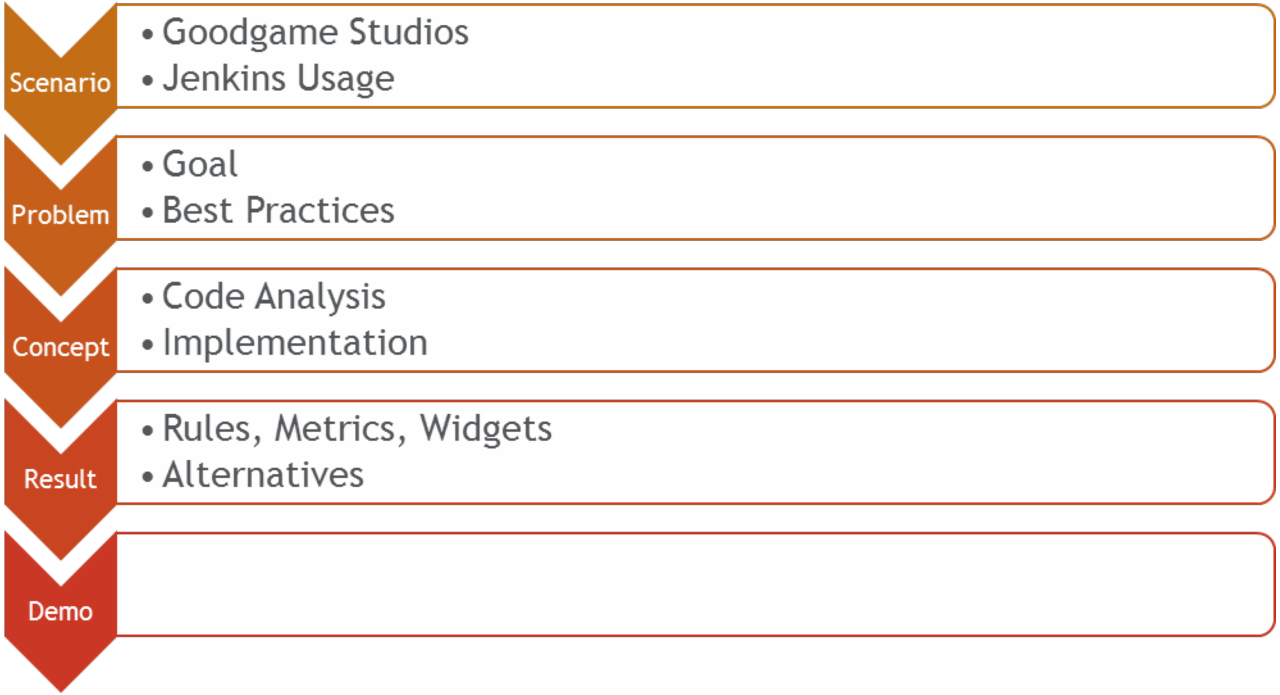
David Hinske

Jenkins World

2016

#JenkinsWorld

Agenda



Scenario
- Goodgame Studios
- Jenkins Usage

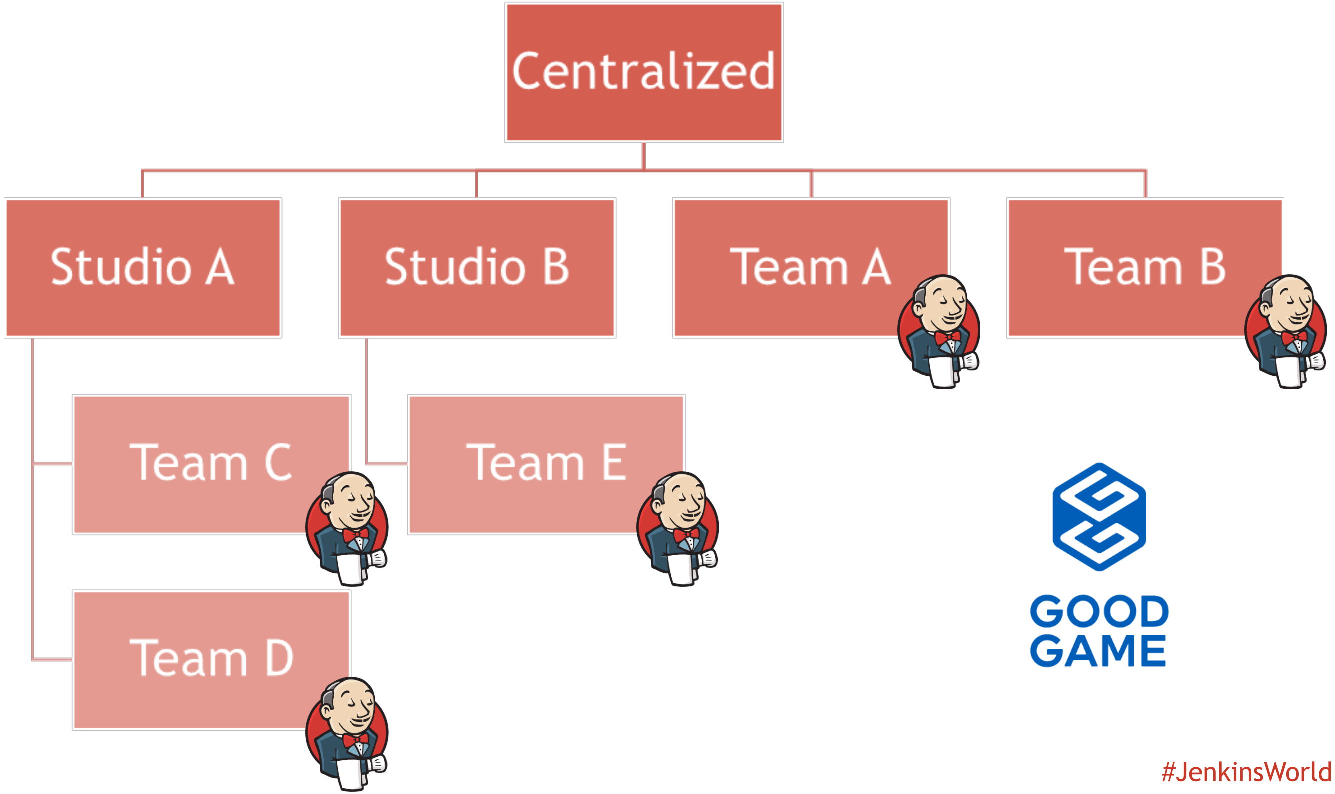
Problem
- Goal
- Best Practices

Concept
- Code Analysis
- Implementation

Result
- Rules, Metrics, Widgets
- Alternatives

Demo

Goodgame Studios



Centralized

Studio A

Studio B

Team A

Team B

Team C

Team D

Team E

GOOD GAME

The Problem

- Very small centralized team
- Huge amount of stakeholders
- Ensure/Support Jenkins health
- Establish standards

➡ Enforcing best practices

Best Practices

- Keep track of provided instances
- Build pipeline = Set of plugins with certain configuration / Pipeline / Other
- Focus on Plugins
  - Usage
  - Configuration
  - Combination
- Keep it simple
- Keep it clean
- Push/Use standard solutions
- Detect possible weaknesses and mailfunctions

Code analysis

- Meet mandatory requirements
- Really understand your application
- Code simplification and sanitizing
- Identifying and fixing potential vulnerabilities, bugs and security threats
- Checking to see if your code complies with best practices and coding standards
- Detect errors in your code before someone else finds them
- Code documentation
- Improve application performance
- Better resource utilization
- It is good practice and your clients will appreciate it

http://www.fasooblog.com/top-10-reasons-why-you-should-use-static-code-analysis

Code analysis

- Meet mandatory requirements
- Really understand your pipeline
- Job-Configuration simplification and sanitizing
- Identifying and fixing potential vulnerabilities, bugs and security threats
- Checking to see if your job-configuration complies with best practices and configuration standards
- Detect errors in your job-configuration before someone else finds them
- Job-Configuration documentation
- Improve pipeline performance
- Better resource utilization
- It is good practice and your clients will appreciate it

http://www.fasooblog.com/top-10-reasons-why-you-should-use-static-code-analysis

Sonarqube

- Software quality management platform
- Rules, Metrics, Widgets, Dashboards,Timelines, Alerts, Cross-Project-Comparison, Extensible
- Adresses 7 axes of code quality
  - Coding standards
  - Potential bugs
  - Documentation & Comments
  - Duplicated Code
  - Complexity
  - Test coverage
  - Design & Architecture

Continuous Inspection
sonarqube

Implementation

What

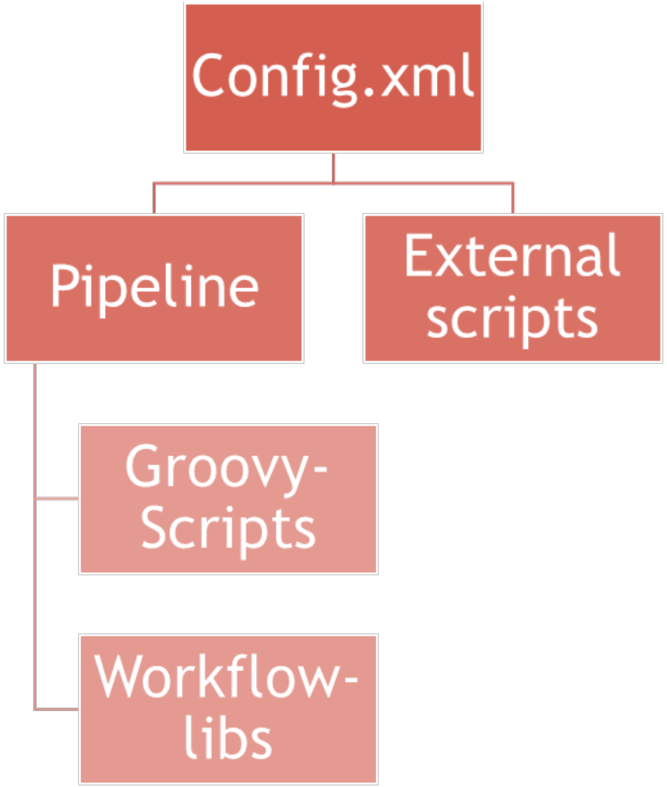
Where

How

Why

Implementation

- ${JENKINS_HOME}
  - /jobs
  - /workflowLibs
  - /workspace



Implementation

What
Where
How
Why

Jenkins World
2016

Sonar

#JenkinsWorld

Implementation

Sonar

Language

```
String KEY
getFileSuffixes()
```

Implementation

What › **Where** › How › Why

Jenkins World
2016

Sonar

Language

Quality Profile

Implementation

What › **Where** › How › Why

Sonar

Language

Quality Profile

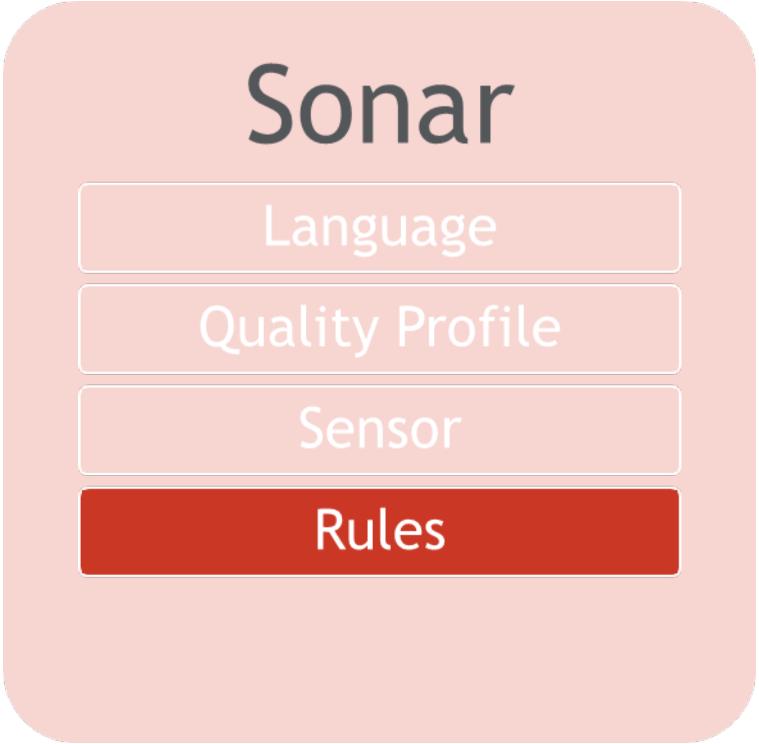
Sensor

```
analyze(...) {
    foreach file
        sort file
    foreach metric
        collectmetrics(jobs)
    foreach jobs
        foreach rule
             validate
}
```

```
JobConfig {
     String name
     ConfigXml configXml
     Pipeline pipeline
     Set<Groovy> groovyScripts
}
```

Implementation

What › Where › **How** › Why



```
validate(JobConfig) {...}
createViolation(file, loc, message)
```

Example

```
<triggers>
   <hudson.triggers.SCMTrigger>
      <spec>H/5 * * * *</spec>
   </hudson.triggers.SCMTrigger>
</triggers>
```

Implementation

Sonar

- Language
- Quality Profile
- Sensor
- Rules
- **Metrics**

```
new Metric.Builder(String key, String
value, Metric.ValueType)
```

Example

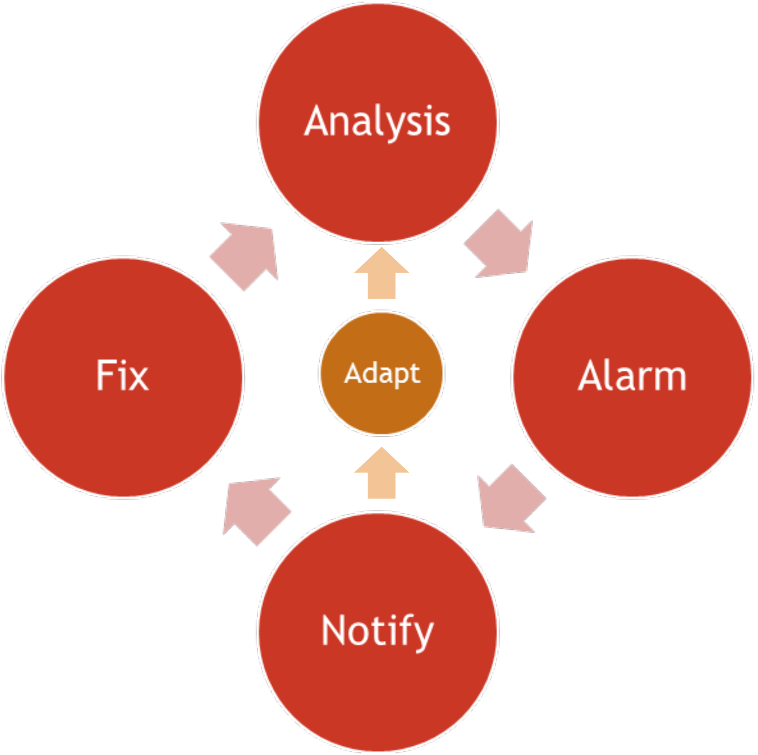
```
AMOUNT_FREESTYLE =
   new Metric.Builder(
      "amount_freestyle",
      "Number of Freestyle-Jobs found",
      Metric.ValueType.INT)


Sensor.analyse() {
   ...
   new Measure(AMOUNT_FREESTYLE);
   measure.setValue(amount_freestyle);
}
```

Improvement Cycle

What → Where → How → **Why**

Jenkins World 2016


Analysis

Alarm

Notify

Fix

Adapt

#JenkinsWorld

Example: Rules

Jenkins World
2016

- Enforce Plugins (Always/Conditional)
- Stop polling, Log Rotator-Usage, Timestamp
- Naming-Convention
  - Scheme
  - No special characters
  - Name-Plugins convention
- Dont use System.Exit(0) in Groovy-Scripts
- ‚H‘ in Cron-Usage
- Distributed Builds
  - Dont build on the master
  - Use labels for slaves
- Pipeline: Check size, Enforce stages/Checkpoints

Example: Metrics

- Job-Types
- Repository-Usage
- Job-Cycle detection
- Cron statistics
- Amount polling/trigger
- Complexity

Challenges

- Different versions of Plugins
- Different ways of implementation
- Different ways of configuration
- Include global configuration
- Include builds and their results

https://github.com/dhinske/sonar-jenkins-plugin

Demo

# Jenkins World

2016

#JenkinsWorld