Patterns to Introduce Continuous Integration to Organizations

Kenichiro Ota
Shift inc.
Tokyo
Japan
oota_ken@hotmail.com
kenichiro.ota@shiftinc.jp

Hiroko Tamagawa
Shift inc.
Tokyo
Japan
hiroko.tamagawa@shiftinc.jp

Lei Wang
Shift inc.
Tokyo
Japan
lei.wang@shiftinc.jp

ABSTRACT

Continuous Integration (CI) is to incrementally integrate software development work within a team by executing a build process whenever changes of work occur. This build process includes compilation, automated testing, static analysis and finally deployment.

From our experiences of introducing the concept of CI to several organizations applying to different projects, we noticed common successful patterns and anti-patterns, which are further organized into a pattern catalog. This paper uses pattern languages to describe the relationship of those patterns.

We believe that people in such roles as team leaders, software configuration management engineers, test automation engineers and process improvement engineers can benefit from our pattern catalog when employing CI in their organizations.

Categories and Subject Descriptors

Continuous Integration

Agile Development

DevOps

Test Automation

General Terms

Development Process

Automation

Verification

1. INTRODUCTION

As agile development methodology prevails in contemporary software development practices [1], CI, one of the key concepts in agile development practice has become prominent because it can greatly reinforce software quality. But if organizations or teams have no experiences of automations in CI, they may soon face anti-patterns and fail the same way as dealing with automations in other domains.

Coming along with CI, we will introduce several automations that are static analysis, code coverage analysis, test automation, packaging deployment and provisioning for test environment all together which will be more difficult than just one singular automation such as test automation though the subjects in both cases are similar.

This pattern catalog explains typical anti-patterns that could happen in the introduction of CI to organizations. Followed by identification and resolution of anti-patterns, we explain patterns that should be adopted.

We list categories of patterns by grouping them, and draw a pattern map that describes the relationship of the patterns.

2. Pattern Category List

We classify patterns into management, introduction and operation categories.

Table 1. Pattern Category List

	Anti-patterns	Patterns
Management	Over expectation of stakeholders	Temporary expedient and effective expedient Involve architect Local CI
Introduction	Automated test first Automate all	Procedure first Write tests for new code Static analysis for testability
Operation	Over dependency on CI Server	Use existing codes as baseline Operation without CI server

Table 2. Category Description

Management	Patterns of managing expectations and requirements of stakeholder when introducing CI
Introduction	Patterns of priorities and scopes when introducing CI
Operation	Patterns of scopes and metrics of operation of CI

3. Pattern Map

Pattern map describes relationship and orders of patterns. The simplex arrow shows the order to adopt patterns: from an anti-pattern to a pattern, we can adopt a pattern after we identify and resolve its corresponding anti-pattern. Lines without arrows are not directional, meaning there are feedbacks between two patterns.

It is recommended to start with the identification of **Automated test first** anti-pattern though the automation of testing process is a benefit from CI. In this case one should start to resolve **Automated tests first** anti-pattern.

A better approach to demonstrating the advantages of CI is to adopt **Temporary expedient and effective expedient** pattern and then to share the results with stakeholders.

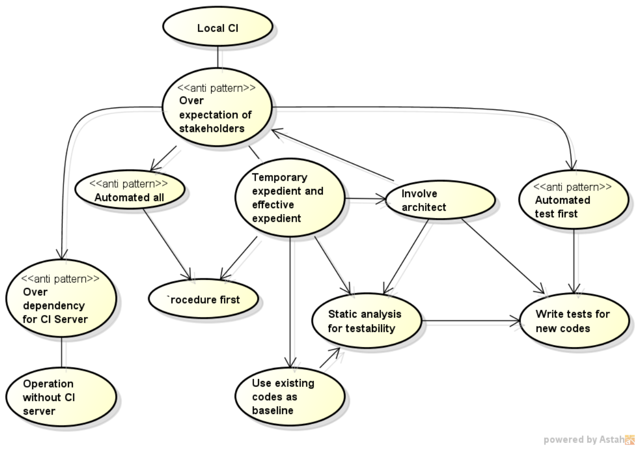
Local CI
<<anti pattern>> Over expectation of stakeholders
<<anti pattern>> Automated all
Temporary expedient and effective expedient
Involve architect
<<anti pattern>> Automated test first
<<anti pattern>> Over dependency for CI Server
`rocedure first
Static analysis for testability
Write tests for new codes
Operation without CI server
Use existing codes as baseline
powered by Astah

Figure 1. Pattern map of patterns to introduce CI

4. Pattern template

4.1 Template for patterns

We use the following templates for patterns.

Name	Coined word that explain pattern name properly
Context	Context that we adopt this pattern
Problem	Problem that we must resolve
Solution	Solution for the problem
Result	Situation that we can resolve the problem
Cause	Cause that we select the solution

4.2 Template for anti-patterns

We use the following templates for anti-patterns. These templates are from "AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis "[2].

Name	Coined word that explain the anti-pattern
Background	Process that the anti-pattern occurs
Symptom and consequences	Negative results if the ant-pattern is adopted
Anecdotal evidence	Conversations being heard in context of the anti-pattern
Typical causes	Typical causes of the anti-pattern
Refactored patterns	Refactored patterns that can resolve the anti-pattern
Refactored solution	Specific solution for the anti-pattern
Related anti-patterns	Any other anti-patterns that are related to this one

5. Anti-Patterns

In the introduction of CI, we sometimes have to confront anti-patterns before we can adopt proper patterns. Therefore describing those potential anti-patterns are necessary.

5.1 Management anti-patterns

5.1.1 Over expectation of stakeholders

Name	Over expectation of stakeholders
Background	Investigation when introducing CI
Symptom and consequences	If adopting **Automated test first** anti-pattern, one starts test automation first and only focuses on it, then the introduction of CI fails. If adopting **Automate all** anti-pattern, one automates all build process. Cost of maintenance and operations will greatly increase.
Anecdotal evidence	If we introduce CI, we will be able to automate tests. If we install CI servers, then all development process will be automated.
Typical causes	Lack of understanding of CI among stakeholders Lack of breakdown and priority management for tasks that we can do in order to introduce CI
Refactored patterns	**Temporary expedient and effective expedient** **Involve architect**
Refactored solution	If adopting **Temporary expedient and effective expedient** pattern, one can classify build process to temporary expedient and effective expedient. Tasks of temporary expedient including compilation and deployment can take effects immediately. Tasks of effective expedient including test automation and use of metrics require process improvement. One can breakdown tasks of temporary expedient and effective expedient and calculate cost and profit. Thus the results of breakdown and the orders of priority of the tasks can be shared with stakeholders. If adopting effective expedient, process and product improvements are needed. Therefore the **Involve architect** pattern is also required, which basically asks to

	work with the architect and improve process and product together.
Related anti-patterns	Automated all

5.2 Introduction anti-patterns

5.2.1 *Automated test first*

Name	Automated test first
Background	Investigation when introducing CI
Symptom and consequences	When trying to automate test first in the environment where there are no automated tests, the introduction of CI fails.
Anecdotal evidence	If we introduce CI, we will be able to automate tests.
Typical causes	Lack of understanding of the position of test automation in CI
Refactored patterns	**Temporary expedient and effective expedient** **Static analysis for testability** **Write tests for new codes** **Involve architect**
Refactored solution	It is commonly perceived that CI and test automation are the necessary pair and the most effective automation within CI is test automation. However test automation is not temporary expedient but effective expedient that requires process improvement. Therefore test automation can only be achieved after **Temporary expedient and effective expedient** pattern is adopted and the order of priority is clarified. If trying to automate tests for legacy code that has no previously automated tests, we adopt **Static analysis for testability** pattern and **Write tests for new code** pattern for new code to improve members' skills and product testability. If product testability is low, adopting **Involve architect** pattern is necessary to change the architecture and design of the product.
Related anti-patterns	Automated all

5.2.2 *Automate all*

Name	Automate all
Background	Introduction of CI
Symptom and consequences	If trying to automate all tasks including testing and maintenance, cost will be greatly increased and the execution of automated tests will be time consuming.
Anecdotal evidence	We should automate all tasks including GUI testing We can see results of static analysis and metrics from a CI server too.
Typical causes	Lack of analysis of cost and effect for tasks that can be accomplished by CI
Refactored patterns	**Procedure first** **Involve architect**
Refactored solution	Although having enough skills, server capacity and capability to automate all, rapidly increased cost of maintenance and execution occurs. One can adopt **Procedure first** pattern and automate only the tasks of temporary expedient. If improving testability by change of architecture and design is needed, one can adopt **Involve architect** pattern to cooperate with the architect to automate tests.
Related anti-patterns	Automated test first

5.3 Operation anti-patterns

5.3.1 *Over dependency on CI Server*

Name	Over dependency on CI Server
Background	Operation of CI
Symptom and consequences	Build process can be automated, but if the CI server stops, nothing can be built anymore. Even worse, build with manual or semi-automated process cannot be done as well.
Anecdotal evidence	As our CI server stops, we can build nothing.
Typical causes	Over dependency on CI servers Lack of understanding of the responsibilities of the necessary tools
Refactored patterns	**Operation without CI Server**
Refactored solution	Over dependency on the CI server should be avoided. Otherwise the CI server will be the single point of failure during the build process. Its risk is same

	as manual build or specific machine build. It is suggested to adopt **Operation without CI Server** pattern, by separating responsibilities between CI servers and other tools like building tools or testing tools. A CI server should be the hub that links all the other tools together.
Related anti-patterns	Automated all

6. Patterns

After the above anti-patterns are identified, we can start to draw and adopt patterns to solve the corresponding problems. Some patterns are particularly aiming to resolve a particular anti-pattern, and if that is the case, we also mention that anti-pattern in the form.

6.1 Management Patterns

6.1.1 Temporary expedient and effective expedient

Name	**Temporary expedient and effective expedient**
Context	Automated build tasks in CI include automated testing, deployment and provisioning, static analysis, and metrics measurement.
Problem	The first automated task needs to be decided
Solution	Tasks in CI are classified into temporary expedient and effective expedient. Temporary expedient doesn't have to improve development process or product, while effective expedient needs. Which to choose depends on the requirements from stakeholders. If stakeholders want immediate outputs, one can adopt **Procedure first** pattern, and the tasks of temporary expedient can be firstly attempted. If stakeholders agree on the introduction of CI for a long-term improvement, One could firstly estimate ROI and then adopt **Write tests for new code** pattern, thus try the tasks of effective expedient.
Result	One can clarify the first automated task. One can avoid **Automated test first** anti-pattern. One can avoid **Over expectation of stakeholders** anti-pattern.
Cause	Automated tasks in CI have two types: tasks of temporary expedient and tasks of effective expedient. Temporary expedient automates tasks that we already execute many times manually including compilation and deployment. Its cost effectiveness is low, but it is easy to try and its benefit of saving time is clear. Tasks of effective expedient require process and product improvement including automated testing and use of metrics.

6.1.2 Involve architect

Name	Involve architect
Context	Trying test automation and use of metrics, which are the tasks of effective expedient, have been decided.
Problem	Because of low testability of the product, it is difficult to automate tests.
Solution	It is required to involve the architect who is the original or current designer by explaining the need of product improvement and profit after the improvement. One can change architecture and design with the architect. Working with the architect one can develop template code of test scripts which can be used within the team to facilitate test automation easily. If the architect is skeptical to test automation, one can adopt **Write tests for new code** pattern and automate tests for utility modules that are independent to the existing architecture.
Result	One can try the tasks of effective expedient that needs the change of architecture and design. Therefore CI adoption can be promoted.
Cause	Testability is low because of the architecture and design of product. Thus to improve testability one needs to change architecture and design. If it is the case, one needs an agreement from the original designers and team. As the change of architecture and design affects productivity of the team, the engineer that the entire team relies on should lead it.

6.1.3 Local CI

Name	Local CI
Context	We are investigating CI. But we can't see what we can do.
Problem	How to find out automated tasks that the team can introduce and work on.
Solution	One can create CI servers and examine feasibility in the local environment. If trying to change the design and source code of the product, a local branch can be created in the version control system. With provisioning tools one can automate the configuration of the CI servers that is for deployment for the team. If under examination the architecture and design

	are required to change, adopting **Involve architect** pattern is suggested.
Result	One can find out tasks to do for the team. Because the configuration of CI servers can be automated, one can complete deployment tasks fairly quickly.
Cause	Because of cost restriction it is difficult to set up physical servers on which investigation can be performed. Creating virtual servers in local environment is easy due to rapid evolution of virtualization technology. Using provisioning tools one can copy the configurations of CI servers that are created on a virtual server.

6.2 Introduction Patterns

6.2.1 Procedure first

Name	Procedure first
Context	Although stakeholders agree on the introduction of CI, they expect immediate outputs.
Problem	The most effective task that has high profit in a short period of time is expected.
Solution	One can start procedure tasks. Procedure tasks include compilation, deployment and packaging. Procedure tasks are tasks that we run manually or semi-automatically with many times. Automation for procedure tasks doesn't have to change architecture and design of the product. So particular members can do the work and the team workload will be low.
Result	One can have outputs in a short period of time. One can prepare for the tasks of effective expedient.
Cause	Tasks of effective expedient are of high cost and take a long time to take effects. Tasks of effective expedient need to change development process and need all members to get involved. Tasks of effective expedient include procedure tasks such as compilation, deployment and packaging. As procedure tasks are tasks that we run manually or semi-automatically with many times, we can automate them in a short period of time thus save execution time.

6.2.2 Write tests for new code

Name	Write tests for new code
Context	Trying test automation in the introduction of CI.
Problem	Need to identify the areas of the product in which test automation can be introduced.
Solution	One can create automated tests for newly added product code. One can create automated tests for utility classes in newly added code. Utility classes are stateless and independent to architecture and design of the product. If can adopt **Involve architect** pattern, one can change architecture and design of the product for testability.
Result	One can automate tests without changing the existing architecture and design.
Cause	If for current product code there are no automated tests, the testability of the code must be low. In this case it is difficult to automate tests for the existing code. Therefore the architecture and design of the product need to change in order to improve the testability. But it is very hard.

6.2.3 Static analysis for testability

Name	Static analysis for testability
Context	When trying test automation in the introduction of CI.
Problem	Member's skills are not enough for test automation
Solution	One can use static analysis tools and metrics measurement tools to improve testability. As static analysis tools and metrics measurement tools can detect code that is nested deeply or high coupling, from which team members can learn how to write testable code. As rules of static analysis tools are in large quantity, we must select essential rules that can effectively help improve the testability and detect critical bugs. One should run those essential rules on CI servers and create reports from static analysis. After team members acquire enough skills to write testable codes, **Write tests for new code** pattern can be adopted.
Result	Team members can acquire enough skills to write testable codes. Team members can adopt **Write tests for new codes** pattern.
Cause	If trying to adopt **Write tests for new codes** pattern, developers must know how to write testable codes. It is very difficult to write automated tests for the

	code with low testability. Static analysis tools can check testability of code.

6.3 Operation Patterns

6.3.1 Use existing codes as baseline

Name	Use existing codes as baseline
Context	Stakeholders request to use existing code on CI.
Problem	When trying to use existing codes on CI, one does not possess finance and skill to do tasks of effective expedient like test automation.
Solution	One can use static analysis report, code metrics and code coverage of existing code as baseline. Existing code does not need to be corrected but used as baseline by which the quality of new code can be measured.
Result	One can fulfil stakeholders' request of using existing code as baseline. Existing code can be used not to increase workload of team members.
Cause	It takes time and cost to adopt effective expedient for existing code. It is easy to run static analysis tools or metrics measurement tools for existing code. Static analysis tools or metrics measurement tools can identify many potential issues.

6.3.2 Operation without CI Server

Name	Operation without CI Server
Context	Running so many tasks like compilation, deployment, static analysis, metrics measurement and provisioning on the CI server.
Problem	Even if CI servers have stopped, build process still needs to continue.
Solution	One should automate or semi-automate build tasks without CI servers. A CI server is the hub to run some automated tasks. But one still can run other tasks from build tools. One can manage configurations of build tasks on version management systems. Thus they can be reverted whenever necessary.
Result	Even if CI servers have stopped, one can still build in our local environment. Even if the configurations of build tasks become invalid, one can still revert them easily.
Cause	If depending on CI servers heavily and when CI servers have stopped, build processes become discontinued. One should be able to build manually or semi-automatically even if CI servers have stopped. Version management of CI servers is not enough comparing to version management system. One should manage all configurations of build tasks in a version management system. One should use tools for the objectives.

7. Relationship with other CI patterns and Best Practice

There exist many literatures discussing patterns and anti-patterns about CI in which ”Continuous Integration Patterns and Anti-patterns”[3] describes a lot of patterns and anti-patterns especially for CI operations.

Moreover, ”Automation for the people: Continuous Integration anti-patterns”[4] explains operational anti-patterns in greater depth.

“Practices of Continuous Integration”[5] describes best practices of CI including its introduction. Best practices are not patterns, but if the best practices have valid reasons, they can be treated as patterns.

We describe relationship between our patterns and those patterns as following.

7.1.1 Local CI and Private Workspace

We can start our investigation for CI by adopting **Private Workspace** pattern from ”Continuous Integration Patterns and Anti-patterns”[2]. **Private Workspace** pattern prevents our investigation effecting mainstream tasks.

7.1.2 Operation without CI Server, Minimal Dependencies and Single Command

We should keep minimal dependencies not only on CI servers, but also on pre-installed tools.

Minimal Dependencies pattern from ”Continuous Integration Patterns and Anti-patterns”[3] recommends minimal dependencies on pre-installed tools.

Single Command pattern from ”Continuous Integration Patterns and Anti-patterns”[3] recommends using single command on build tasks. We can use **Single Command** pattern to accomplish **Operation without CI Server** pattern.

7.1.3 Use existing codes as baseline and Continuous Inspection

We can use **Continuous Inspection** pattern from ”Continuous Integration Patterns and Anti-patterns”[3] for **Use existing codes as baseline** pattern.

7.1.4 Automate all, Bloated builds delay rapid feedback and Keep the Build Fast

Automate all anti-pattern leads to long build time. If we can resolve **Bloated builds delay rapid feed** anti-pattern from “Automation for the people: Continuous Integration anti-patterns”[4] and **Keep the Build Fast practice** from “Practices of Continuous Integration”[5] by using build pipeline, problem of long build time can be resolved. This is another solution for slow build.

7.1.5 Procedure first and Automate Deployment

Procedure first pattern recommends automating deployment as task of temporary expedient. **Automate deployment** practice from “Practices of Continuous Integration”[5] also asks to automate deployment by other reasons. The reasons are safe deployment and easy rollback. Automated deployment can reduce cost by reducing mistakes of the manual operations.

8. Use cases of Patterns

We below describe some use cases of patterns for some particular situations.

8.1.1 Stakeholders’ expectation is high

Some stakeholders think that if CI is introduced, one can automate everything. By identifying **Automate all** and **Over expectation of stakeholders** anti-patterns, one can resolve the issues and adopt the appropriate patterns.

8.1.2 Stakeholders can’t decide what should start

One can classify tasks in CI into tasks of temporary expedient and effective expedient. Temporary expedient does not have to improve development process or product, but effective expedient does need. Thus one can adopt **Temporary expedient and effective expedient** pattern and reach at the same page with stakeholders on ROI of the tasks.

If one selects temporary expedient as first task, **Procedure first** pattern can be used. One should evaluate automated tasks of procedure by adopting **Local CI** pattern, then spread to the team.

8.1.3 Want to start tasks of effective expedient

As tasks of effective expedient need cooperation of the architect who understands the existing architecture and design of the product, one can adopt **Involve architect** pattern.

Even if one can refactor architecture and design, existing code may not have automated tests and may not be testable. In that case, one can adopt **Use existing code as baseline** pattern and **Write test code for new code** pattern.

8.1.4 Can’t build without CI servers

Often **Over dependency for CI Server** anti-pattern brings challenge, but one can resolve this anti-pattern by adopting **Operation without CI Server** pattern, **Minimal Dependencies** pattern and **Single Command** pattern.

9. ACKNOWLEDGMENTS

We created this pattern catalog from our experience of introducing CI to several organizations for various projects. My appreciation goes to the architects and team members who have inspired us.

And we also would like to thank our colleagues, CI experts and CI community members who reviewed our paper.

Lastly we want to thank the authors who wrote CI patterns such as from “Continuous Integration Patterns and Anti-patterns [2]” for sharing their profound knowledge.

We hope that this pattern catalog can bring beneficial outcomes to team leaders, software configuration management engineers, test automation engineers and process improvement engineers when trying to employ CI within organizations.

10. REFERENCES

[1] State of Agile Report http://stateofagile.versionone.com/

[2] William J. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick, Thomas J. Mowbray. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. ISBN-13: 978-0471197133

[3] Continuous Integration Patterns and Anti-pattern. http://refcardz.dzone.com/refcardz/continuous-integration#refcard-download-social-buttons-display

[4] Automation for the people: Continuous Integration anti-patterns http://www.ibm.com/developerworks/java/library/j-ap11297/

[5] Practices of Continuous Integration http://martinfowler.com/articles/continuousIntegration.html

[6] John Ferguson Smart. Jenkins: The Definitive Guide. O'Reilly Media. ISBN-13: 978-1449305352

[7] Paul M. Duvall, Steve Matyas, Andrew Glover. Continuous Integration: Improving Software Quality and Reducing Risk. ISBN-13: 978-0321336385