An Architecture for Self-Organizing Continuous Delivery Pipelines

Master Thesis – Final Talk

Jan Simon Döring
jan.simon.doering@rwth-aachen.de
Towards Continuous Delivery 2.0
The next generation Software Delivery Systems
Continuous Delivery is a set of principles and practices to reduce the cost, time and risk of delivering incremental changes to users.

Humble, Farley
Current generation Software Delivery Systems

VCS → build → test → release → Cloud

Delivery Model

node {
  ...
  stage('build') {
    sh 'mvn clean package'
  }
  stage('deploy dev') {
    sshagent(['RemoteCredentials']) {
      sh "scp target/*.jar root@:/opt/jenkins-demo.jar"
      sh "ssh root@$server nohup java -Dserver.port=$port -jar /opt/jenkins-demo.jar &"
    }
  }
}

Jenkins 2 – Pipeline as code example
Modeling Problems

„The build scripts are complicated or complex“
Problems, causes and solutions when adopting continuous delivery—A systematic literature review
Laukkanen, Itkonen, Lassenius - Information and Software Technology (Feb. 2017)

A delivery model change causes a failure of the next build with a probability of 40%.
Mining Changes of Build Processes in the Context of Continuous Integration
Benedikt Holmes – Bachelor Thesis RWTH Aachen (2017)
Build Monolith

“The build system cannot be modified flexibly”

New technologies

New process steps

...
Challenges

- **Project Evolution**
  - Technical Evolution (e.g. new tools)
  - Process Evolution (e.g. new process activities)
  - Organizational Evolution (e.g. new policy)

- **Modeling Usability**
  - Minimize required knowledge (technical & process)
  - Support the user

  Be flexible!
  Be maintainable!
  Model Simplification!
  Provide Assistance via Tooling!
Domain Driven Design
Tackling Complexity in the Heart of Software
Delivery Process Abstractions

- **Transformation**
  - code → compile → binary

- **Assessment**
  - binary → unit test → report

- **Quality Gate**
  - binary → decision → Promoted binary
  - report → policy

Inspired by J. Hermans
„Evolution of Build Artifacts in Continuous Delivery“
1. **Git checkout**

2. **Static analysis**
   - **Policy**
   - **Transformation**
   - **Assessment**
   - **Quality gate**
   - Options: e.g., # of .java files with LoC > 20
   - **Configuration**

3. **Quality gate**
   - **Compile**
   - **Transform**
   - **Assess**
   - **Report**
Domain Driven Design – Core Domain

![Diagram of Delivery Process and Core Domain components]

- **Delivery Process**
- **Stage**
- **Activity**
- **Logical Dependency**
- **External Delivery Model**
- **Views**
- **Command**
- **Configuration Model**
- **Activity Configuration**
- **Artifact**
- **Execution Result**
- **Assessment**
- **Transformation**
- **Quality Gate**
- **Execution Process Execution**
- **Result Model**

**Relationships:**
- **describes**
- **has**
- **defines**
- **dependent on**
- **produces**
- **configured by**

**Key concepts:**
- Logical Dependency
- Functional Dependency
- Activity Dependency

**Execution Results:**
- Technical Result (EnvInfo, Logs, ..)
- Functional Result (ActivityResult)

**Legend:**
- **Internal Delivery Model**
- **Execution**
- **Reasoning Support**
- **External Delivery Model II**
Central Design Decisions

Conceptual level

Isolation & Autonomy
Standardized Interface

Service Principles

Architecture level

Artifact + Config + Selector

Command

Artificial activity Artifact

Activity - Microservice
Self-Organization

Analyze, generate, adapt and optimize the delivery model by the system itself.
Activity Specification

• Be explicit about input / output schema
• Allows for reasoning

- Required Artifact Types
- Parameter Schema
Self-Organizing

- Idea: Tackle modeling challenge by *automatic planning*
  - Reduce required knowledge
  - Compliance & Best practices
  - Optimization
  - Automatic evolution

- Model *Separation* allows for *custom* (tailored) description languages
Pipeline Description Language

**Stages Definitions**

- **name**: build
- **transformations**:
  - checkout
  - compile
  - jar

**Transformations Definitions**

- **name**: checkout
  - **service**: git-service
  - **activity**: checkout
  - **configuration**:
    - repositoryUri: https://github.com/spring-...
  - name: compile
    - **service**: maven-service
    - **activity**: compile
    - **dependsOn**:
      - alias: repo
      - ref: p://this/transformations/checkout
    - **configuration**:
      - workspace: "@repo"

**Assessments Definitions**

- **name**: unitTests
  - **service**: maven-service
  - **activity**: test
  - **dependsOn**:
    - alias: compile
    - ref: p://this/transformations/compile
    - alias: repo
    - ref: p://this/transformations/checkout
  - **configuration**:
    - pom: "@repo"
    - classes: "@compile"

**Quality Gates Definitions**

- **strategy**: auto
- **policies**:
  - **name**: unitTestPolicy
  - **interpretation**: threshold
  - ref: p://this/assessments/unitTests
  - **actualValue**: passedRate
  - **setPoint**: 1
Planning Operations

- Add / remove / modify stage
- Add / remove / modify activity
- Add dependencies

- name: compile
  service: maven-service
  activity: compile
  dependsOn:
    - alias: repo
      ref: p://this/transformations/checkout/workspace

- name: assemble
  service: maven-service
  activity: assemble
  configuration:
    workspace: "@repo"
    classes: "@compile"
  dependsOn:
    - alias: repo
      ref: p://this/transformations/checkout/workspace
    - alias: compile
      ref: p://this/transformations/compile/classes
Architecture Dynamics

[Diagram of Architecture Dynamics]

**Delivery System Management Tool**

- **Queries**
  - View Service

**Event Store**

- **Event Bus**
  - Events
  - Model Events
  - Delivery Process Events
  - Execution Events

**Model Service**

- **External Delivery Model**

**Process Planner**

- **Delivery Process**

**Orchestrator**

- **Delivery Process Execution**

**Control Flow** ➔ **Data Flow**

- **Import phase of Architecture Control Flow**
- **Planning phase of Architecture Control Flow**
- **Execution phase of Architecture Control Flow**
Core Delivery Framework

- **Import**: M2M from external to internal delivery model
- **Planning**: Adapt & Optimize model
  - Model-based planning: Consider specified activities
  - Project-based planning: additionally use project data, other external sources..
- **Execution**: Perform modeled activities
Activity Service Blueprint
Activity Service SDK

- For Java Spring Services
- Eases development of new activity services
  - Derives specification
  - Registers specification
  - Provides execution API
  - Orchestrator interaction

```java
@PipelineCommand(name = "checkout", type = CommandType.TRANSFORMATION)
public class GitCheckoutCmd implements Command<GitCheckoutResult> {
  private final GitCheckoutProperties properties;
  private final GitExecutor git;

  public GitCheckoutCmd(GitCheckoutProps properties, GitClient git) {
    this.properties = properties;
    this.git = git;
  }

  @Override
  public GitCheckoutResult execute(ExecutionMonitor monitor) {
    //implement me
  }
}
```
<table>
<thead>
<tr>
<th>Service</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmeter-service</td>
<td>2</td>
</tr>
<tr>
<td>git-service</td>
<td>4</td>
</tr>
<tr>
<td>maven-service</td>
<td>5</td>
</tr>
<tr>
<td>docker-service</td>
<td>2</td>
</tr>
</tbody>
</table>
Case Study

• Industry project: IBE Web Service (API Gateway)
  • 65k LoC, ~18 team members

• Tech: Java EE, Maven
• Dependencies: Database, Keycloak
• Tests: Junit, Jmeter

• Objective:
  • Is the core domain applicable?
  • Can new technologies be integrated easily?
  • What is the impact of Self-Organization?
Case Study – Delivery Process

28

Git Checkout → Maven Compile → QS Gate 1 → Maven Package → Docker Build → Docker Provision → Quality Gate 2 → Deploy to Artifactory

Project Sources → Java Classes → Gateway Ear

Maven Test → Surefire Reports → Cobertura Reports

Jmeter Tests → Jmeter Reports

Teardown Docker

Conceptual

New activity

Previously performed manually by web service team

Docker Container Names

SWC Software Construction

RWTH AACHEN UNIVERSITY
Case Study – Simplify model by automatic mapping

Manual Model

- LoC = 124
- “Complexity” = 18

Planned Model

- LoC = 66
- “Complexity” = 1

Complexity = 1 + Number of dependencies (similar to McCabe)
Case Study – Project Planner

- **Maven Planner**
  - Analyzes maven sub-modules
  - Optimization goal: Fail-fast

Execution Speedup: -9.14%

Fails up to 3.31 times faster

Test Failure Seeding
Case Study - Conclusion

• Core domain applicable?
  – 100% coverage of process with concepts
  – No adoptions required

• Can new technologies be integrated easily?
  – We added a Jmeter Service & Docker Service
  – No adoptions to existing services
  – Activity Service SDK

• Impacts of Self-Organization
  – Reduced LoC by ~47%, Less complex
  – Up to 3,31 times faster feedback
  – Stakeholder Feedback: Reduced transparency (“magic”)
Threads to validity

- Complex System (structural)

- Activity Implementation Overhead ("first implement, then use")

  Too much overhead for small projects
Future Work

- Modeling Tools
  - E.g. Graphical or Smart Tooling (auto-completion, recommendation)

- Improve Validation
  - Smells and anti-pattern detection
  - Policy-based

- Extended Planning
  - Dynamic (Re-) Planning
  - More planners & multi-target planning
  - Learning from history

- Delivery Ecosystem
Summary

Challenges

- Project Evolution
  - Technical Evolution (e.g. new tools)
  - Process Evolution (e.g. new process activities)
  - Organizational Evolution (e.g. new non-functional reqs / policies)
- Modeling Usability
  - Minimize required knowledge (technical & process)
  - No assistance
- Be flexible!
- Be maintainable!
- Model Simplification!
- Provide Assistance via Tooling!

Core Delivery Framework

- Import: MQV from external to internal delivery model
- Planning: Adopt & Optimize model
  - Model-based planning; Generate specific activities
  - Project-based planning: Additionally use project data, other external sources...
- Execution: Performs modeled activities

Domain Driven Design – Core Domain

Case Study – Simplify model by automatic mapping

Manual Model

- Leaf = 114
- "Complexity" = 18

Planed Model

- Leaf = 48
- "Complexity" = 5

Complexity = 1 * Number of descendants (similar to McCabe)

Case Study – Project Planner

- Maven Planner
  - Analyze main sub-modules
  - Optimization goal: Fairness
- Execution Speedup: ~9.5X
  - Falls within 5-10 hours faster
What are next generation Software Delivery Systems?

- Easier to use!
- Really flexible!
- Smart!
- Robust!
- Maintainable!