Jan Simon Döring
jan.simon.doering@rwth-aachen.de

An Architecture for Self-Organizing Continuous Delivery Pipelines

Master Thesis – Final Talk

SWC Software Construction | RWTH AACHEN UNIVERSITY

Towards Continuous Delivery 2.0

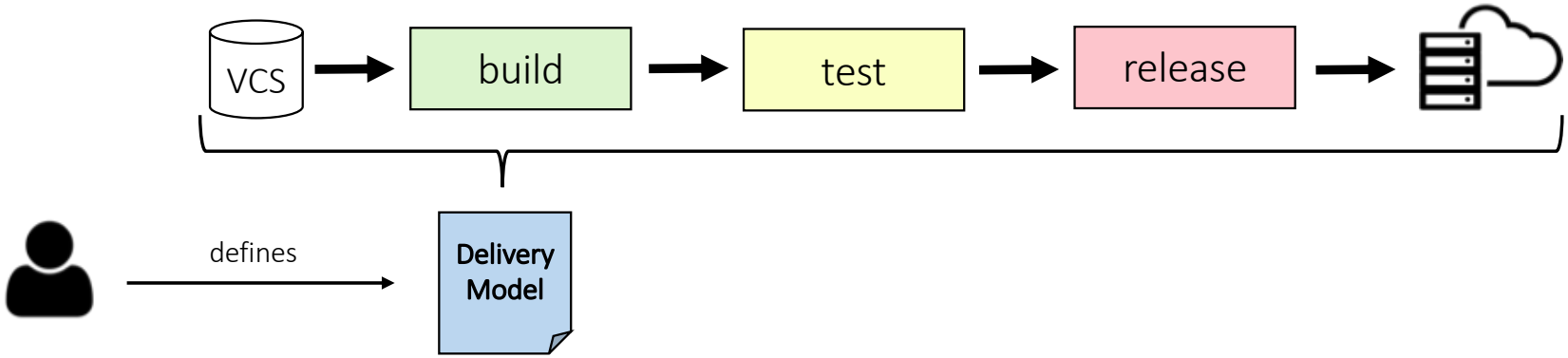
The next generation Software Delivery Systems

Continuous Delivery

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




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

Jenkins 2 – Pipeline as code example

Delivery System

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

...

Laukkanen, Itkonen, Lassenius
"Problems, causes and solutions when adopting continuous delivery—A systematic literature review"
Information and Software Technology (Feb. 2017)

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!

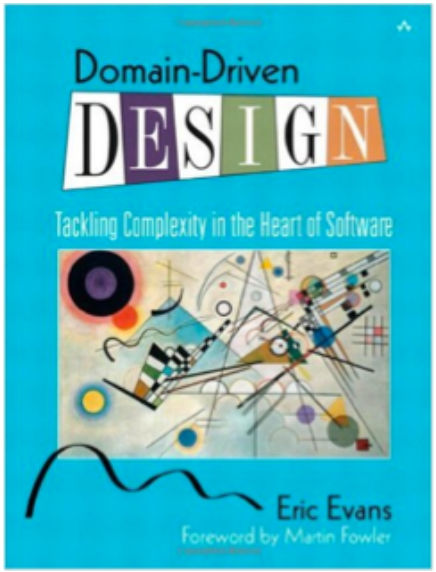
Be maintainable!

Model Simplification!

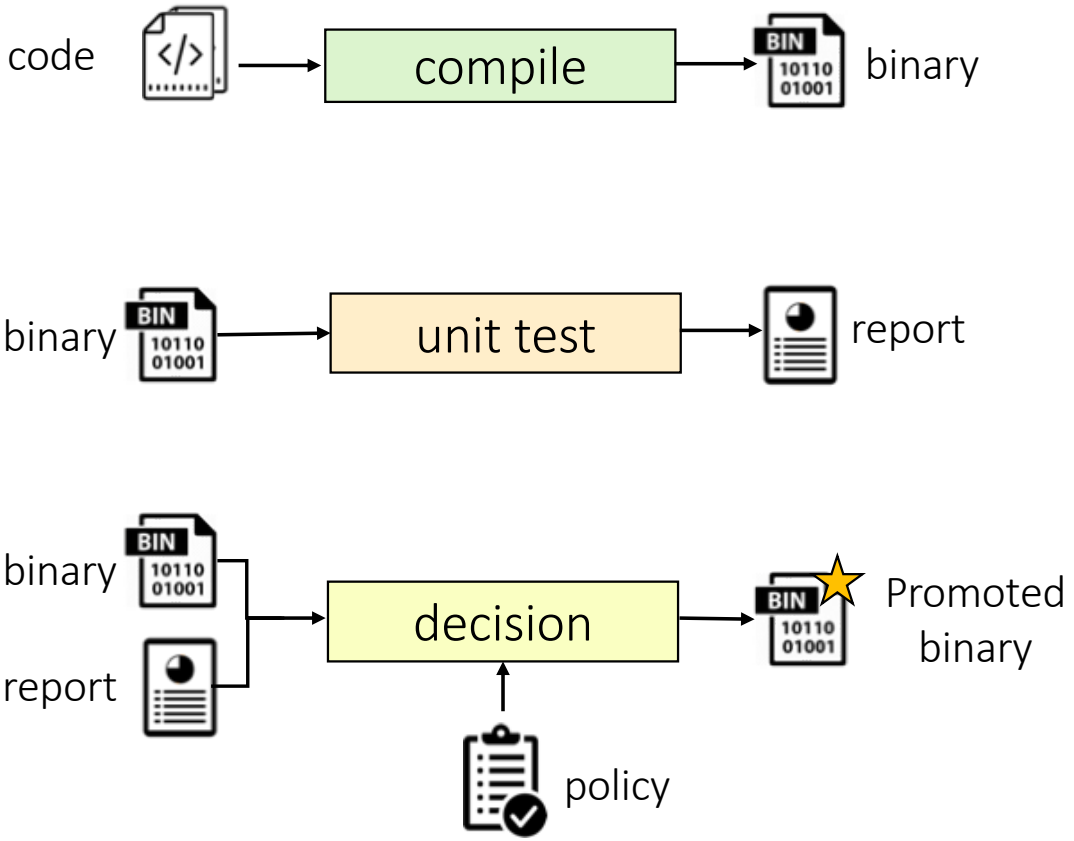
Provide Assistance via Tooling!

Domain Driven Design

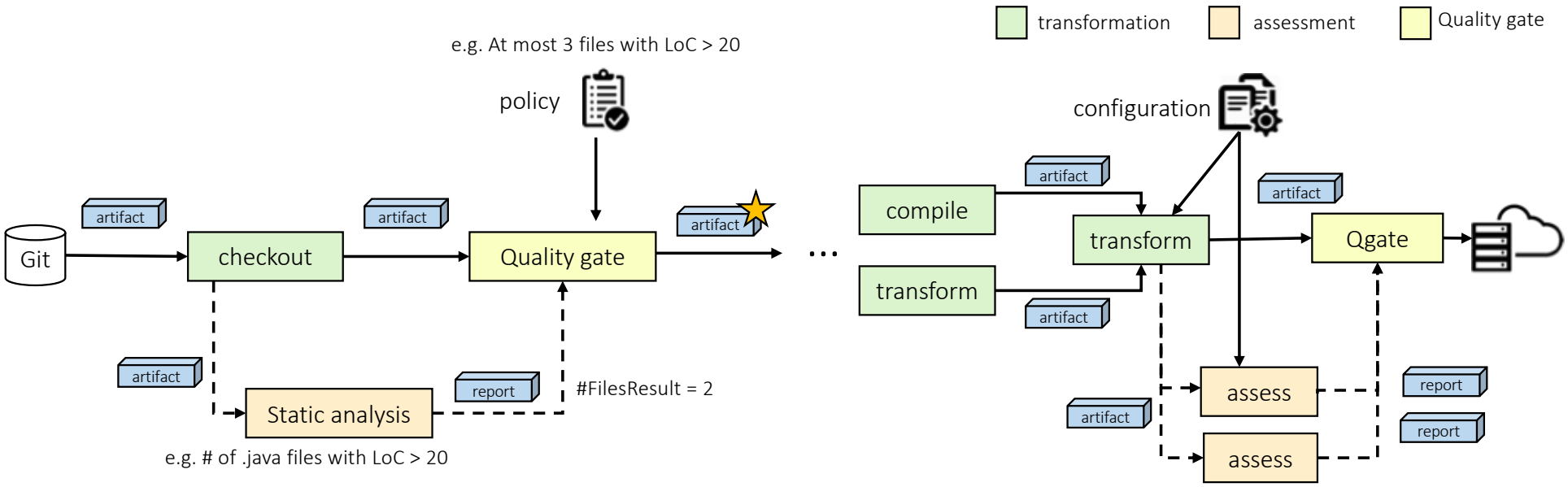
Tackling Complexity in the Heart of Software



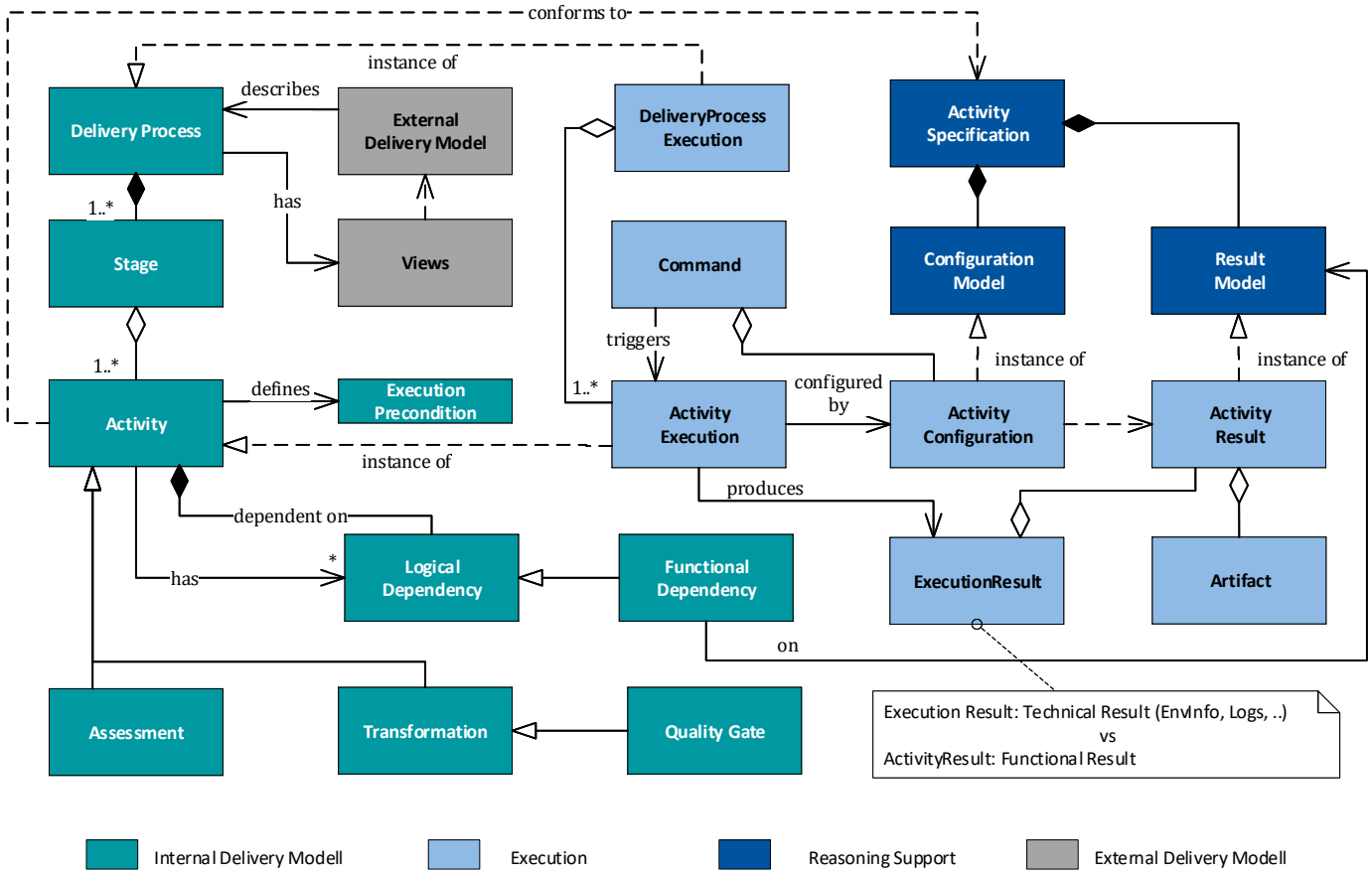
Delivery Process Abstractions



Inspired by J. Hermanns
„Evolution of Build Artifacts in Continuous Delivery“
Bachelor Thesis, RWTH Aachen (2015)

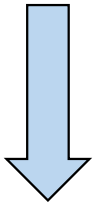

transformation
assessment
Quality gate
e.g. At most 3 files with LoC > 20
policy
configuration
artifact
Git
checkout
Quality gate
compile
transform
transform
Qgate
assess
assess
report
Static analysis
#FilesResult = 2
e.g. # of .java files with LoC > 20

Domain Driven Design – Core Domain



Central Design Decisions

Conceptual level

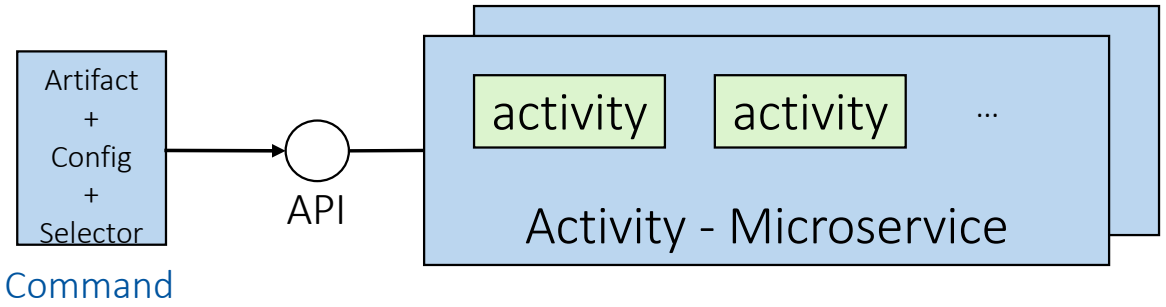


Isolation & Autonomy

Standardized Interface

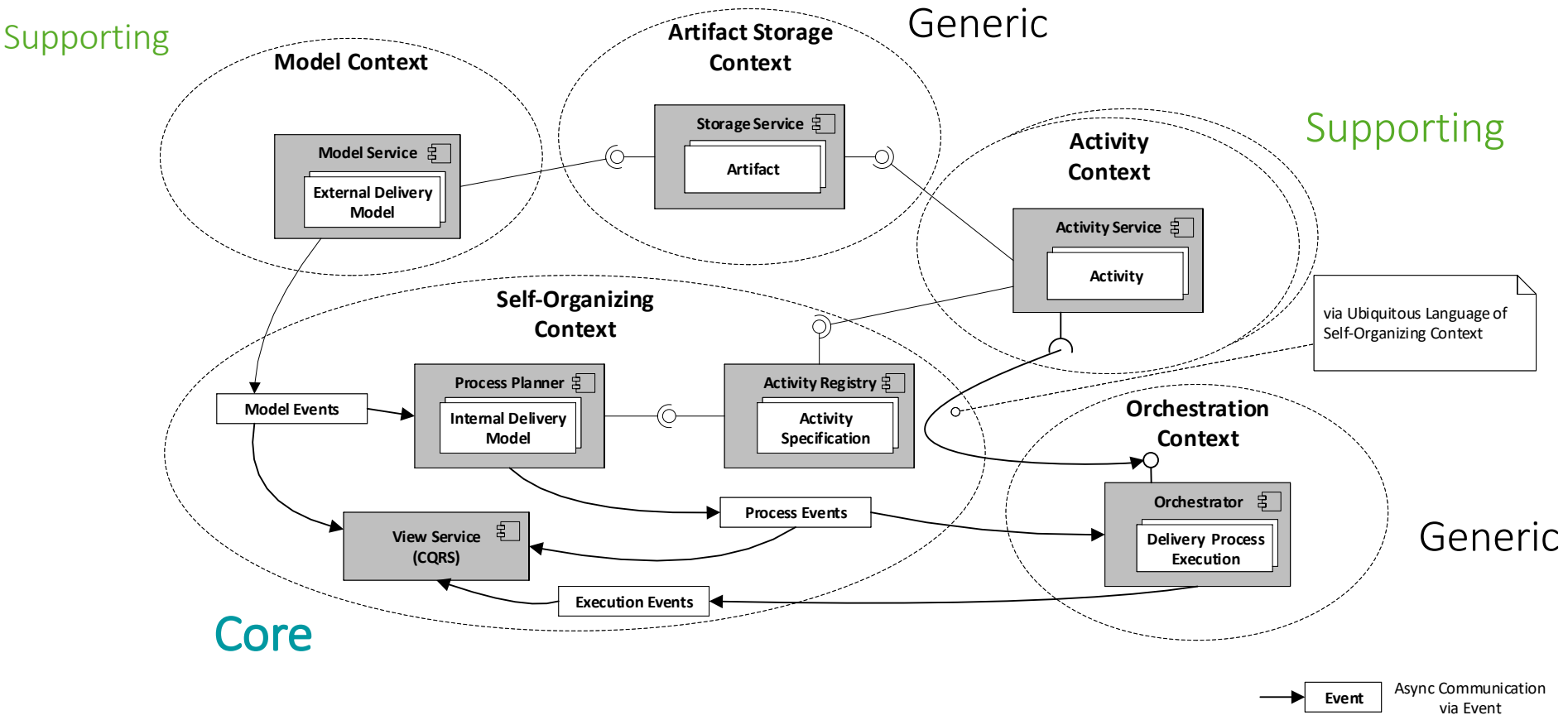
Service Principles

Architecture level



Command

Central Design Decisions

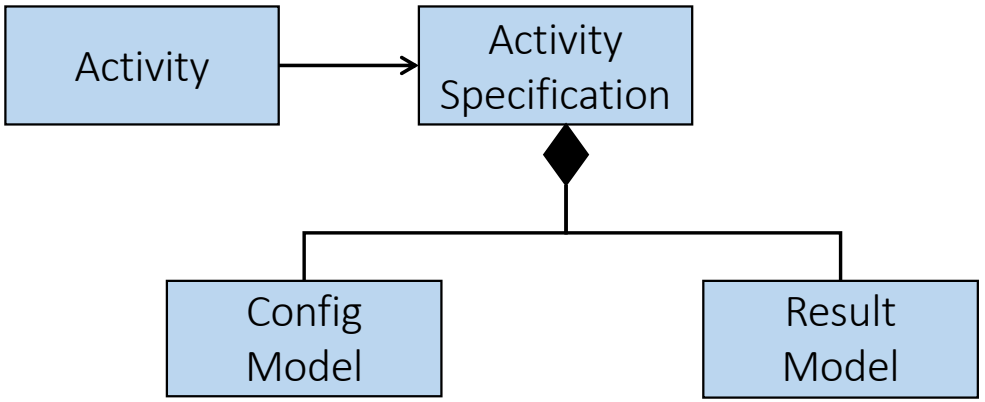


Self-Organization

Analyze, **generate, adapt** and **optimize**
the delivery model by the system itself

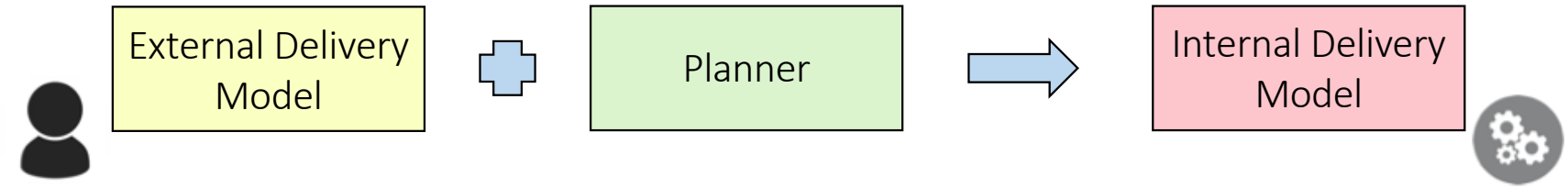
Activity Specification

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



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

Stage Definitions

```
stages:
 - name: build
   transformations:
     - checkout
     - compile
     - jar
```

Transformation Definitions

```
transformations:
 - 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“
```

Assessment Definitions

```
assessments:
 - 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 Gate Definitions

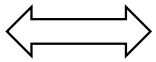
```
qualityGates:
 - strategy: auto
   policies:
     - name: unitTestPolicy
       interpretation: threshold
       ref: p://this/assessments/unitTests
       actualValue: passedRate
       setPoint: 1
```

Planning Operations

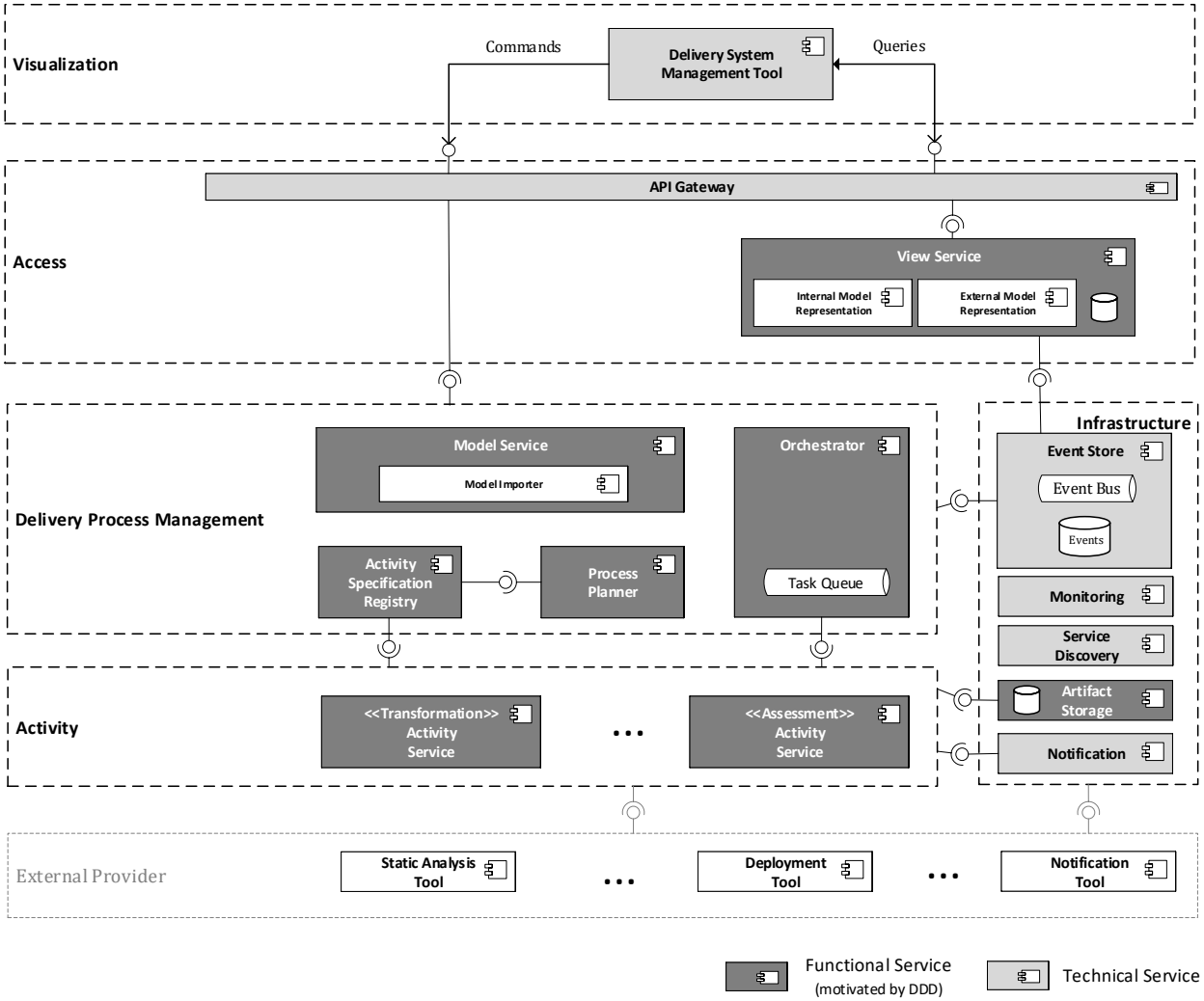
- Add / remove / modify stage
- Add / remove / modify activity
- Add dependencies

```
- name: compile
  service: maven-service
  activity: compile

- name: assemble
  service: maven-service
  activity: assemble
```

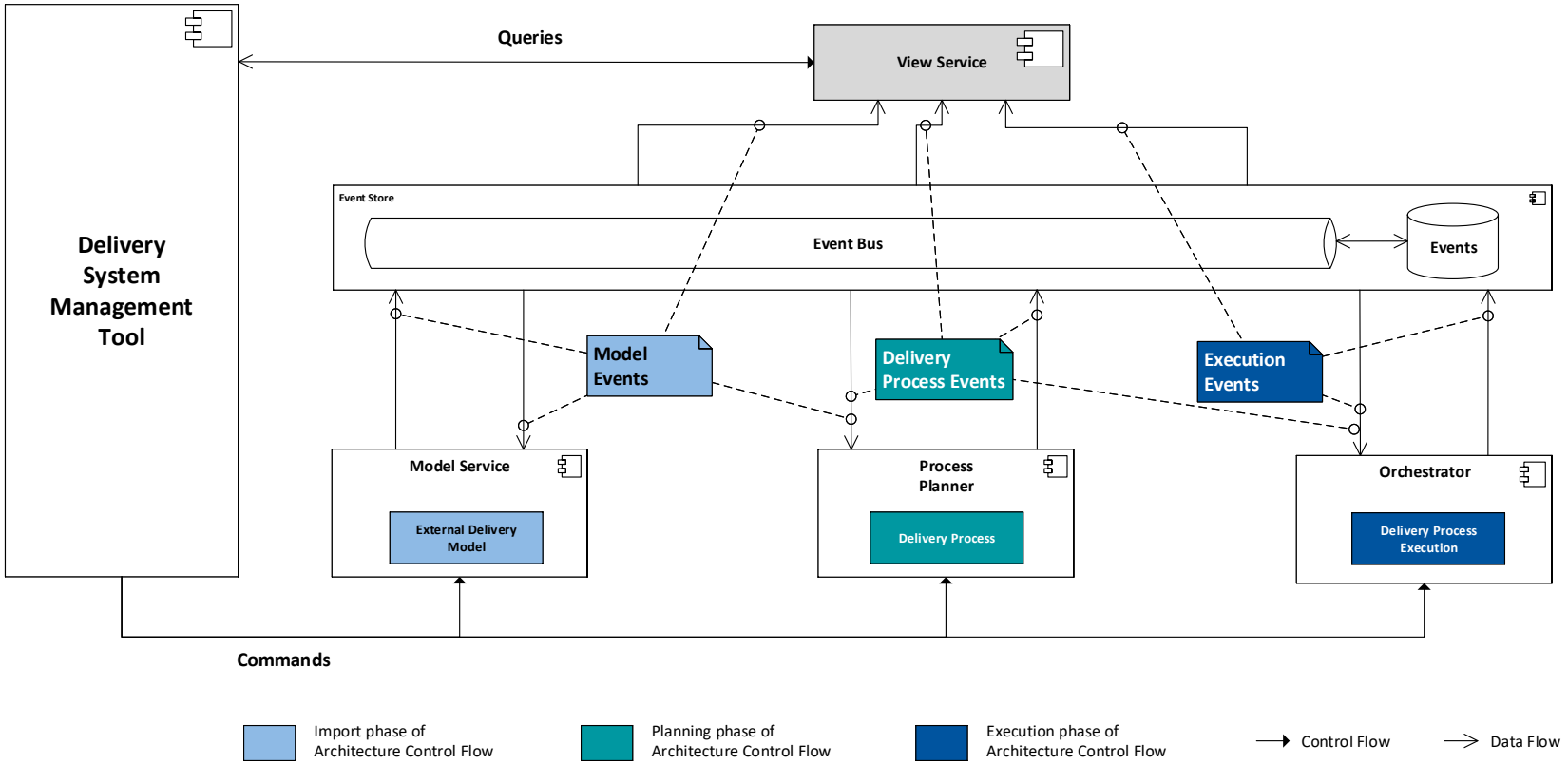
Semantically equivalent



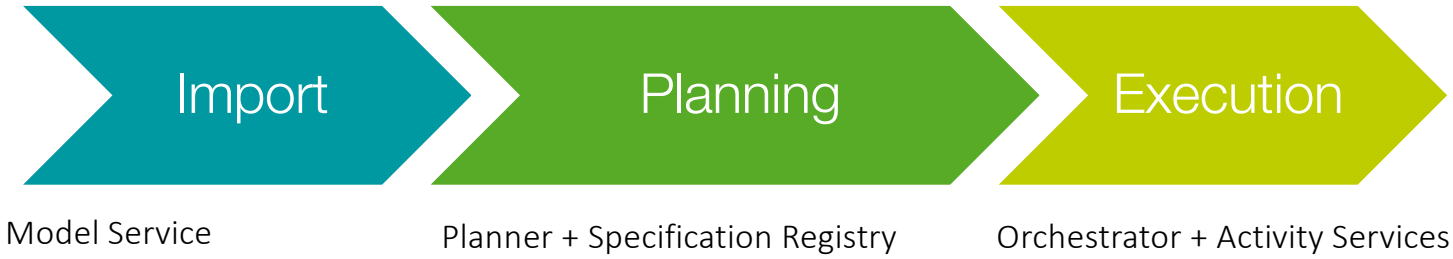
```
- name: compile
  service: maven-service
  activity: compile
  dependsOn:
    - alias: repo
      ref: p://this/transformations/checkout/workspace
  configuration:
    workspace: "@repo"

- 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
```


Visualization
Commands
Delivery System Management Tool
Queries
Access
API Gateway
View Service
Internal Model Representation
External Model Representation
Infrastructure
Delivery Process Management
Model Service
Model Importer
Orchestrator
Event Store
Event Bus
Events
Activity Specification Registry
Process Planner
Task Queue
Monitoring
Service Discovery
Activity
<<Transformation>> Activity Service
<<Assessment>> Activity Service
Artifact Storage
Notification
External Provider
Static Analysis Tool
Deployment Tool
Notification Tool
Functional Service
(motivated by DDD)
Technical Service

Architecture Dynamics

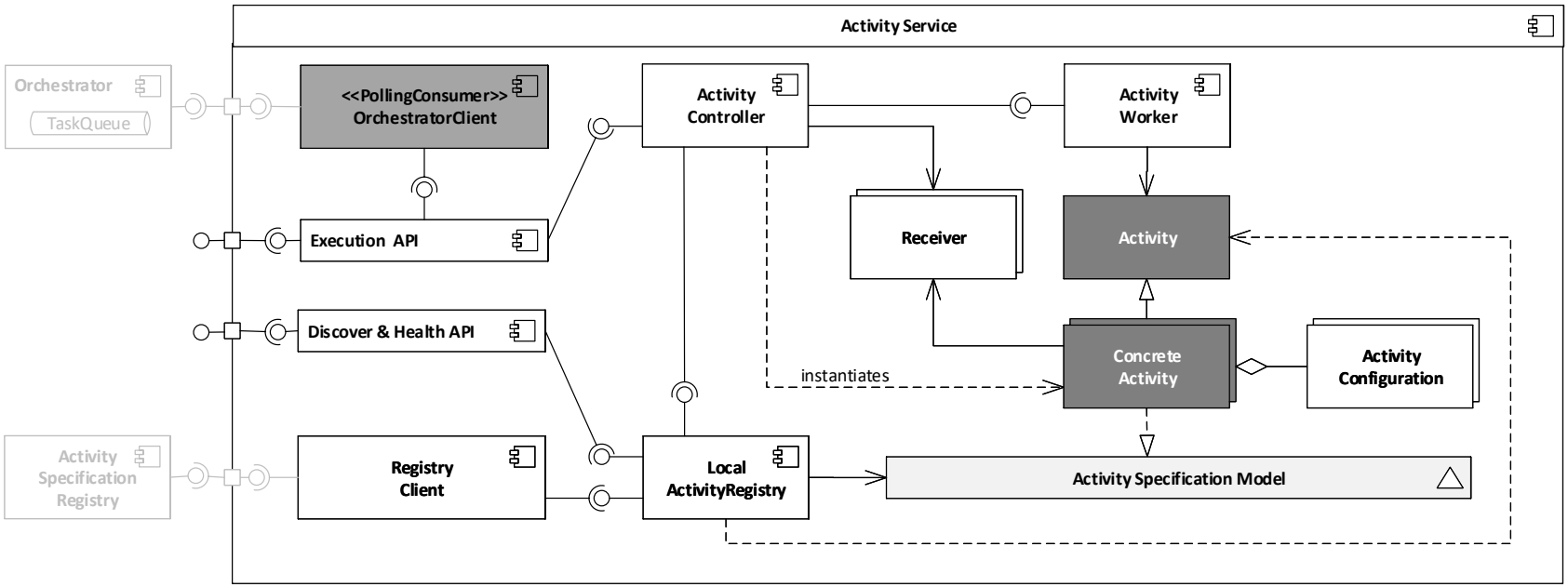


Core Delivery Framework



- **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

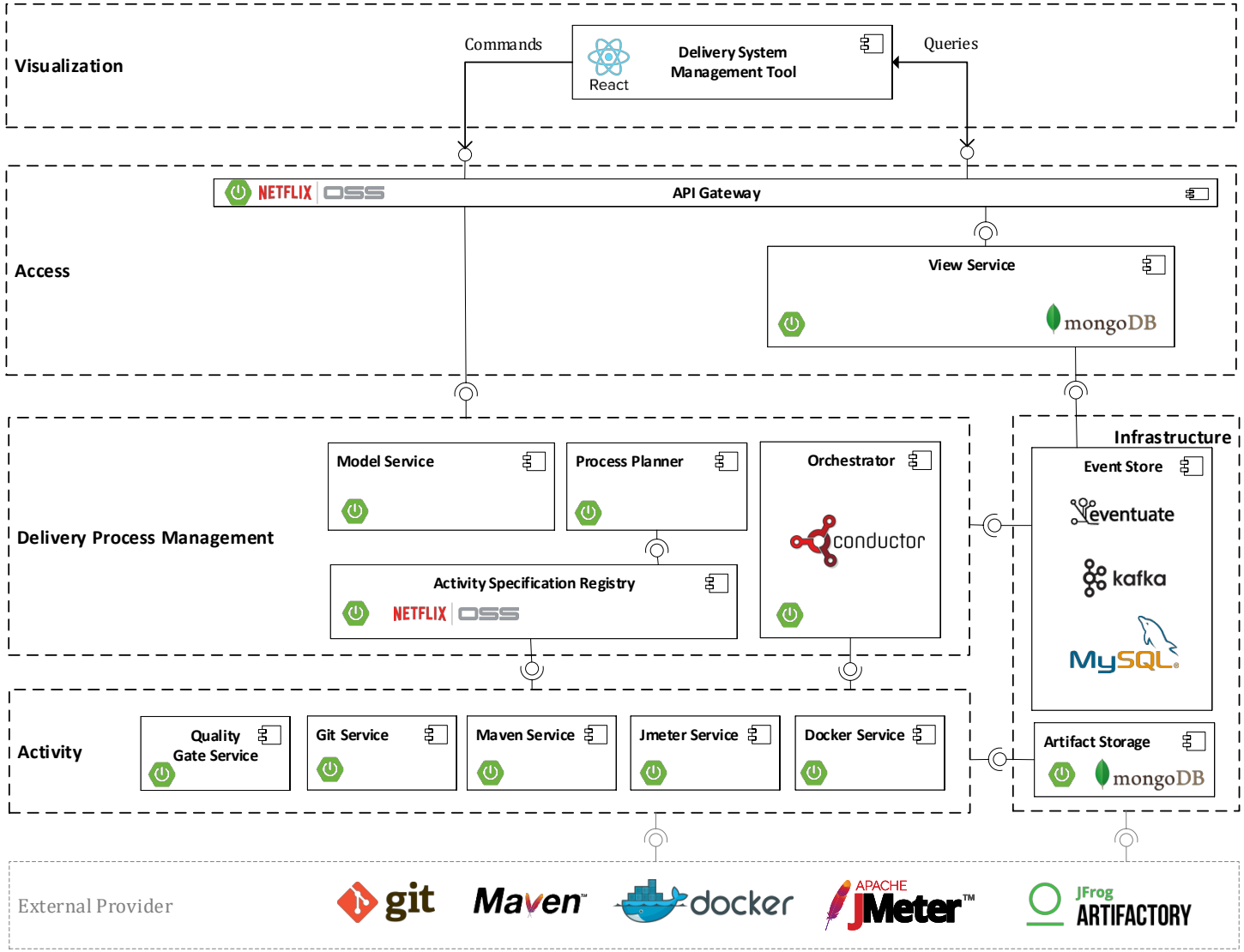
Activity Service Blueprint



Activity Service SDK

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

```
@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
  }
}
```


Visualization
Commands
React
Delivery System Management Tool
Queries
Access
NETFLIX OSS
API Gateway
View Service
mongoDB
Delivery Process Management
Model Service
Process Planner
Orchestrator
conductor
Activity Specification Registry
NETFLIX OSS
Infrastructure
Event Store
eventuate
kafka
MySQL
Activity
Quality Gate Service
Git Service
Maven Service
Jmeter Service
Docker Service
Artifact Storage
mongoDB
External Provider
git
Maven
docker
APACHE JMeter
JFrog ARTIFACTORY


DEMO


PipelineUI
Jan Simon
dockermain/sds/inventory
Delivery System
Inventory
Pipelines
> jmeter-service: 2 Commands
> git-service: 4 Commands
> maven-service: 5 Commands
> docker-service: 2 Commands

Case Study

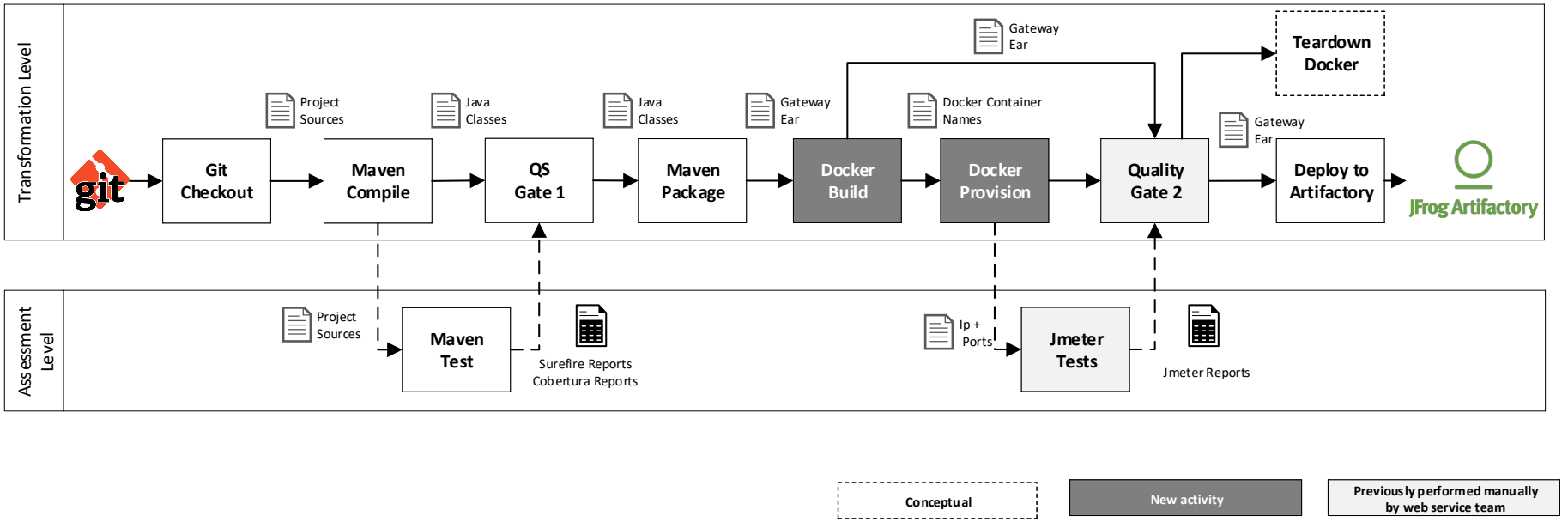
traveltainment
an AMADEUS company

- 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



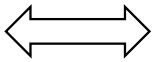
Case Study – Simplify model by automatic mapping

Manual Model

```
stages:
  - name: buildTTGateway
    transformations:
      - checkout
      - compile
      - assemble
  # ...
transformations:
  # ...
  - 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
  # ...
assessments:
  # ...
qualityGates:
  # ...
```

- LoC = 124
- “Complexity” = 18

Semantically equivalent



Planned Model

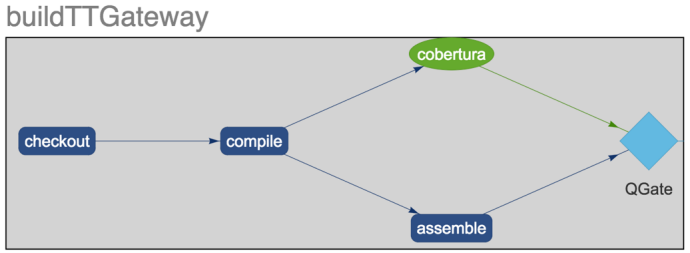
```
stages:
  - name: buildTTGateway
    transformations:
      - checkout
      - compile
      - assemble
  # ...
transformations:
  # ...
  - name: assemble
    service: maven-service
    activity: assemble
  # ...
assessments:
  # ...
qualityGates:
  # ...
```

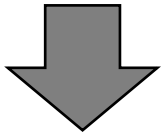
- LoC = 66
- “Complexity” = 1

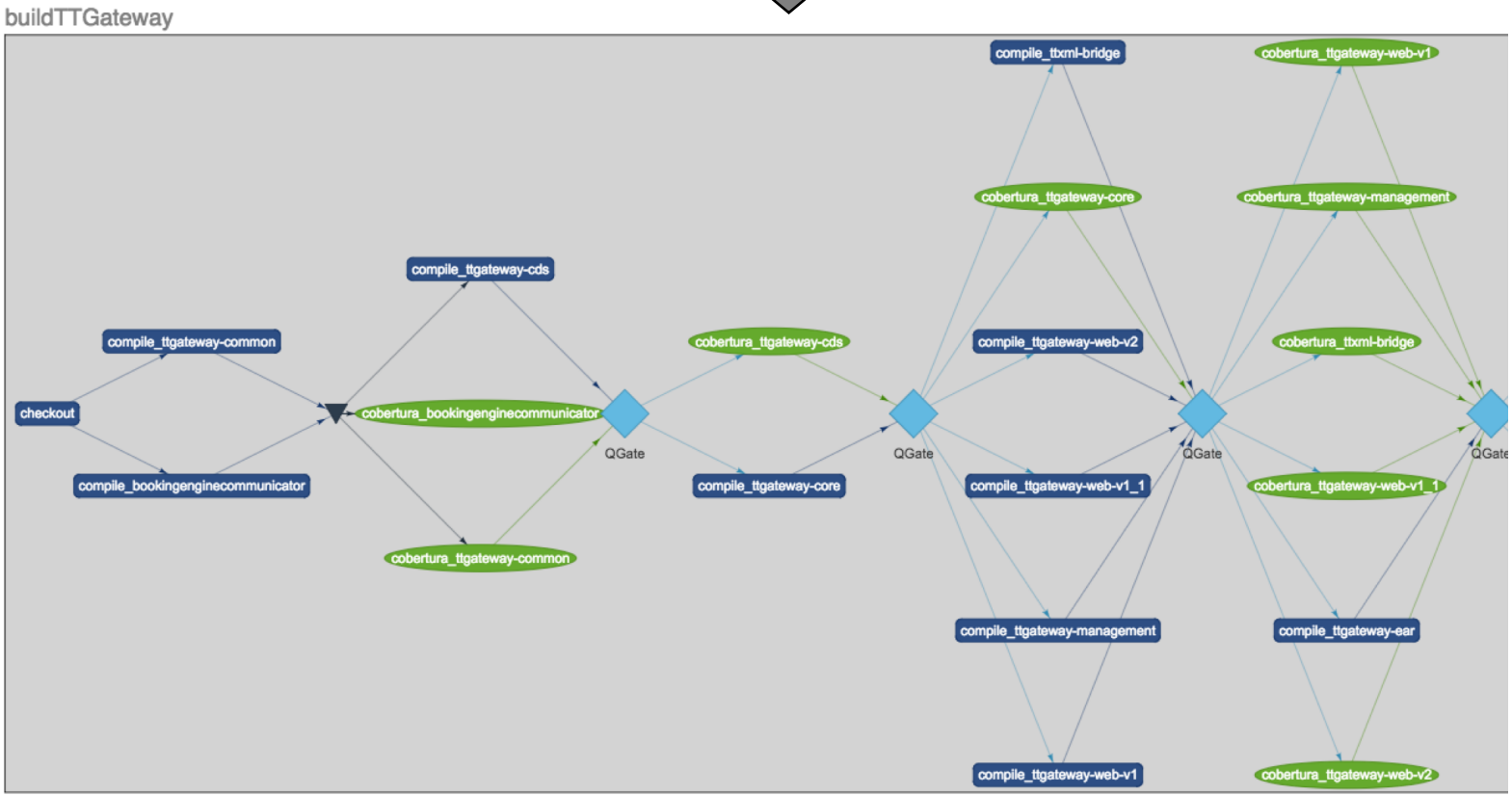
Complexity = 1 + Number of dependencies (similar to McCabe)

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”)

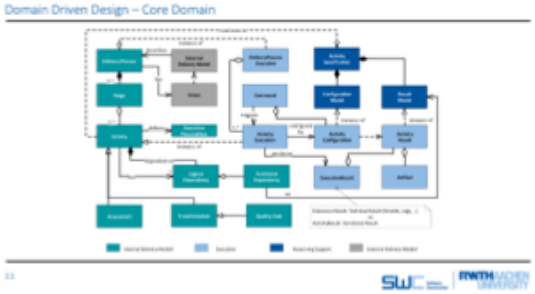
Too much overhead for small projects

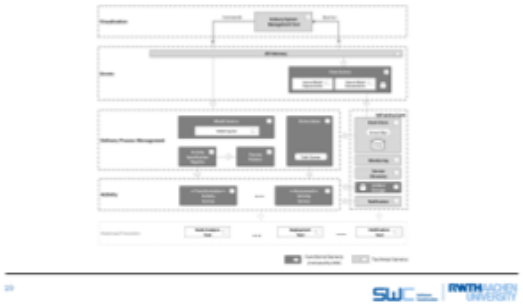
Future Work

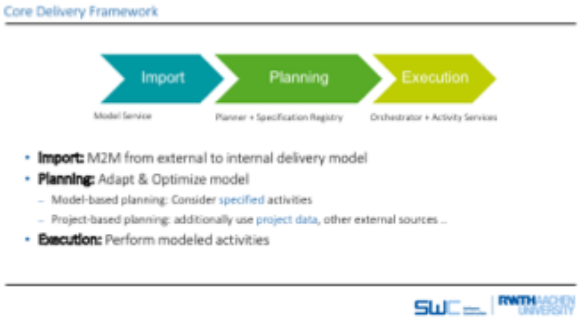
- Modeling Tools
  - E.g. Graphical or Smart Tooling (auto-completion, recommendation)
- Improve Validation
  - Smells and anti-pattern detection
  - Policy-based
- Extended Planning
  - Dynamic (Re-) Planning
  - More planners & multi-target planning
  - Learning from history
- Delivery Ecosystem

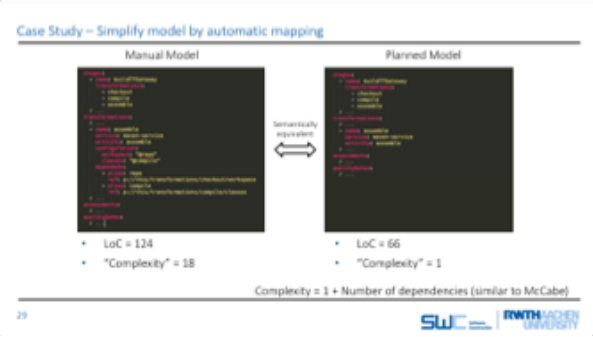
Summary


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


Domain Driven Design – Core Domain




Core Delivery Framework
Import
Planning
Execution
Model Service
Planner + Specification Registry
Orchestrator + Activity Services
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


Case Study – Simplify model by automatic mapping
Manual Model
Planned Model
LoC = 124
"Complexity" = 18
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

What are next generation Software Delivery Systems?

Easier to use!

Really flexible!

Smart!

Robust!

Maintainable!