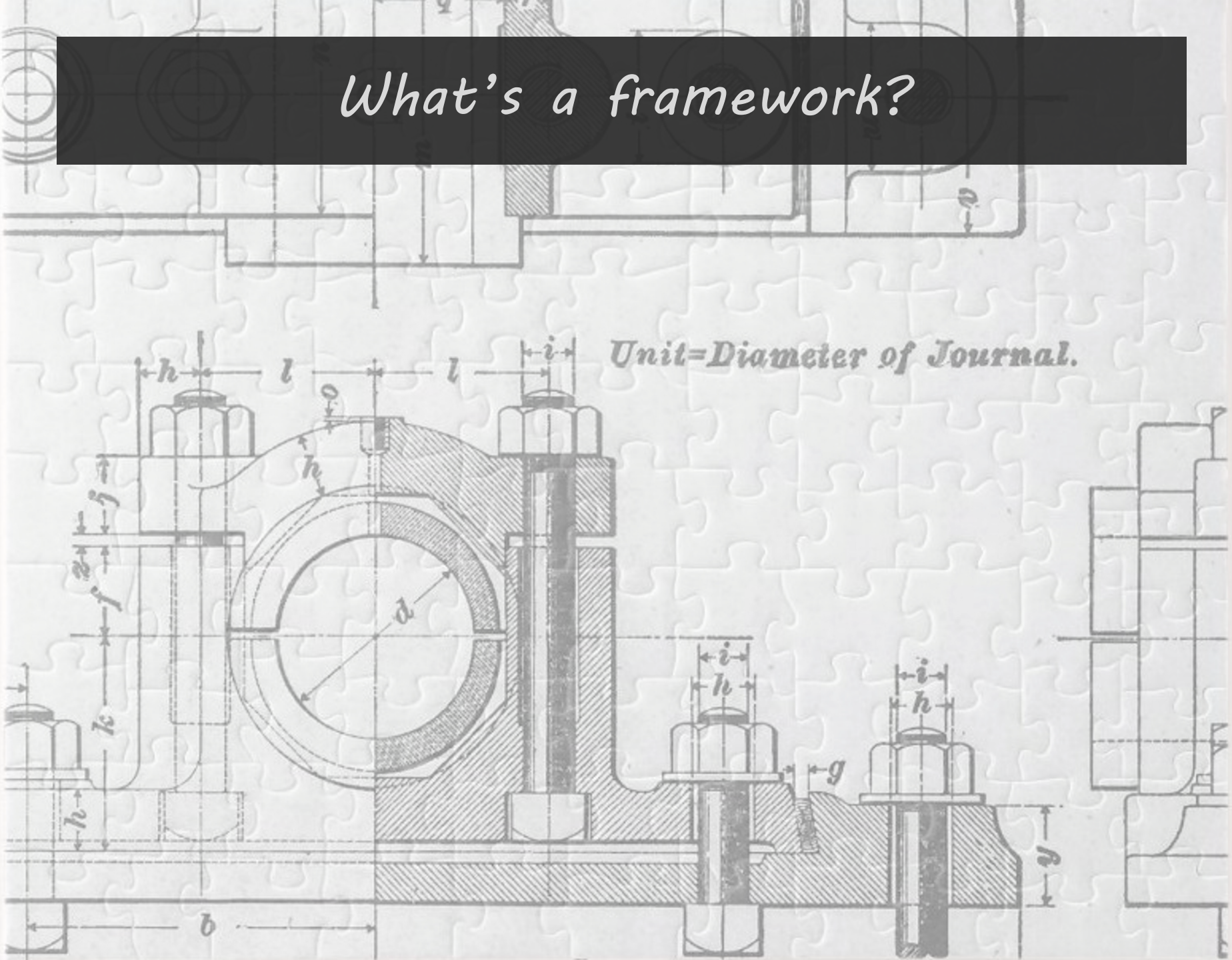
Test Frameworks

Putting together the puzzle pieces

Building & evaluating test automation framework

Aaron Evans
aarone@one-shore.com

What's a framework?



Dictionary Definition

Framework

Article Talk

From Wikipedia, the free encyclopedia

A **framework** is a generic term commonly referring to an essential supporting structure which other things are built on top of.

Framework may refer to:

Computing [edit]

- Application framework, used to implement the structure of an application for an operating system
- Architecture framework
- Content management framework, reusable components of a content management system
- CSS framework
- Enterprise architecture framework
- Framework (office suite), a DOS office application suite in 1984
- Framework Computer, a laptop manufacturer for modular laptops
- Framework-oriented design, uses existing frameworks for application design
- List of rich web application frameworks
- Logical framework
- Multimedia framework, handles media on a computer and through a network
- Software framework, a reusable set of libraries or classes for a software system or subsystem
- Web framework, for development of dynamic websites, web applications, and web services

iety.

My Definition

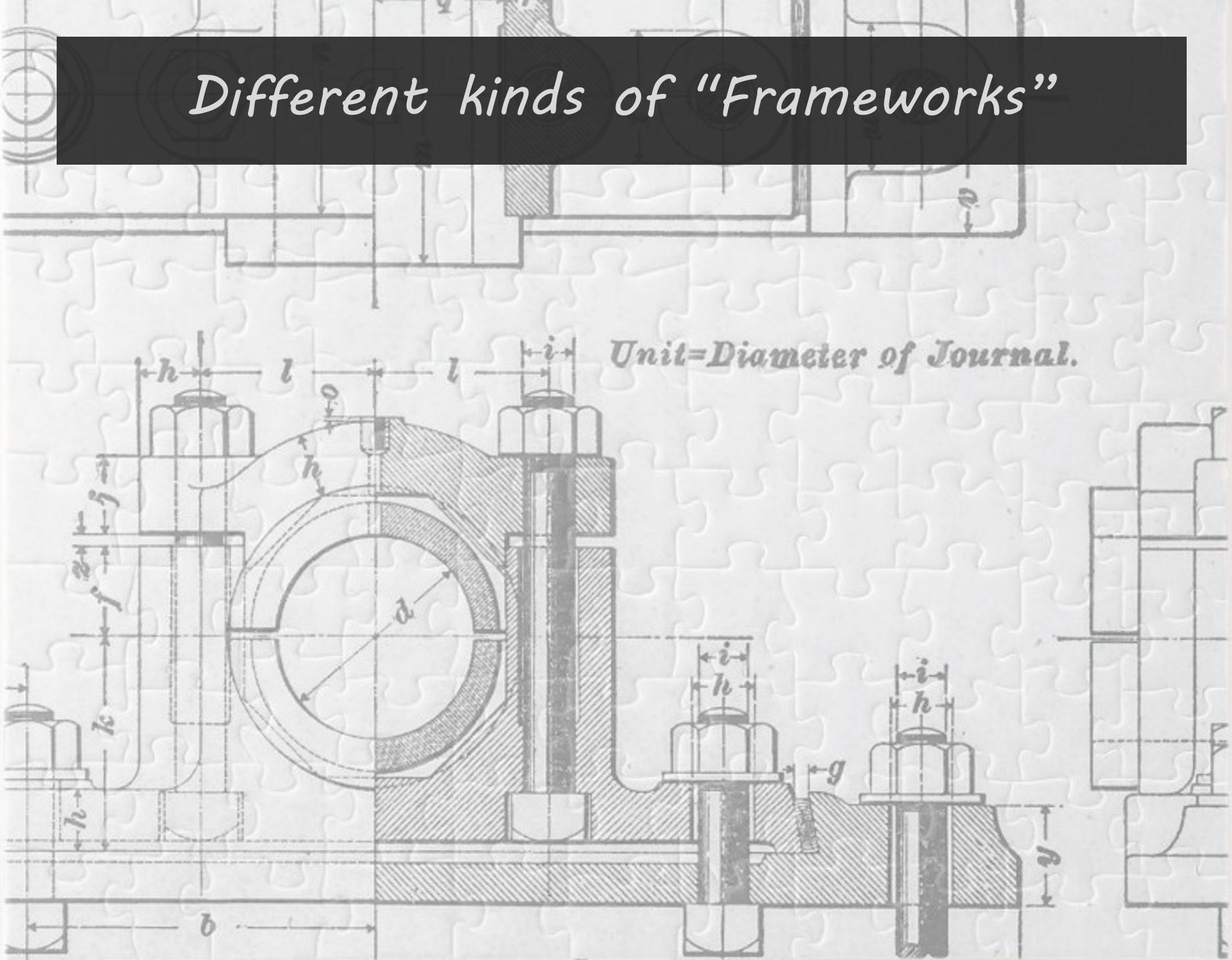
Tools, Processes, Architecture, Patterns

used to

Write, Maintain, Execute, Analyze

test automation

Different kinds of "Frameworks"

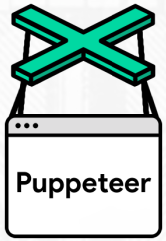


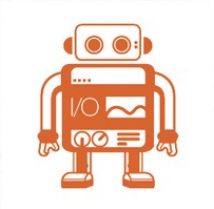
Different kinds of "Frameworks"

Test Runner	JUnit, TestNG, Mocha, PyTest
Automation Library	Selenium, Appium, Cypress, Playwright
Test Specifications	Cucumber, SpecFlow, Robot Framework, Spock
GUI Tools	TestComplete, Tricentis Tosca, UFT/QTP, Katalon Studio
Reporting	Allure Framework, Extent Reports, ReportPortal
CI /CD	Jenkins, Azure DevOps, Github Actions, CircleCI
Cloud Services	Sauce Labs, Browserstack, LambdaTest, Applitools
Code Structure	Serenity, Selenide, Karate
Load/Performance	LoadRunner, JMeter, Locust, Lighthouse
Test Management	TestRail, Zephyr, X-Ray, ALM/Quality Center

Test Runners

Automation Tools & Libraries





Test Specifications

Cucumber

specflow

jbehave

Commercial GUI Automation Tools

Test Structure

API Testing

Load and Performance

Reporting

Test Management

Jira Software

Zephyr

XRAY

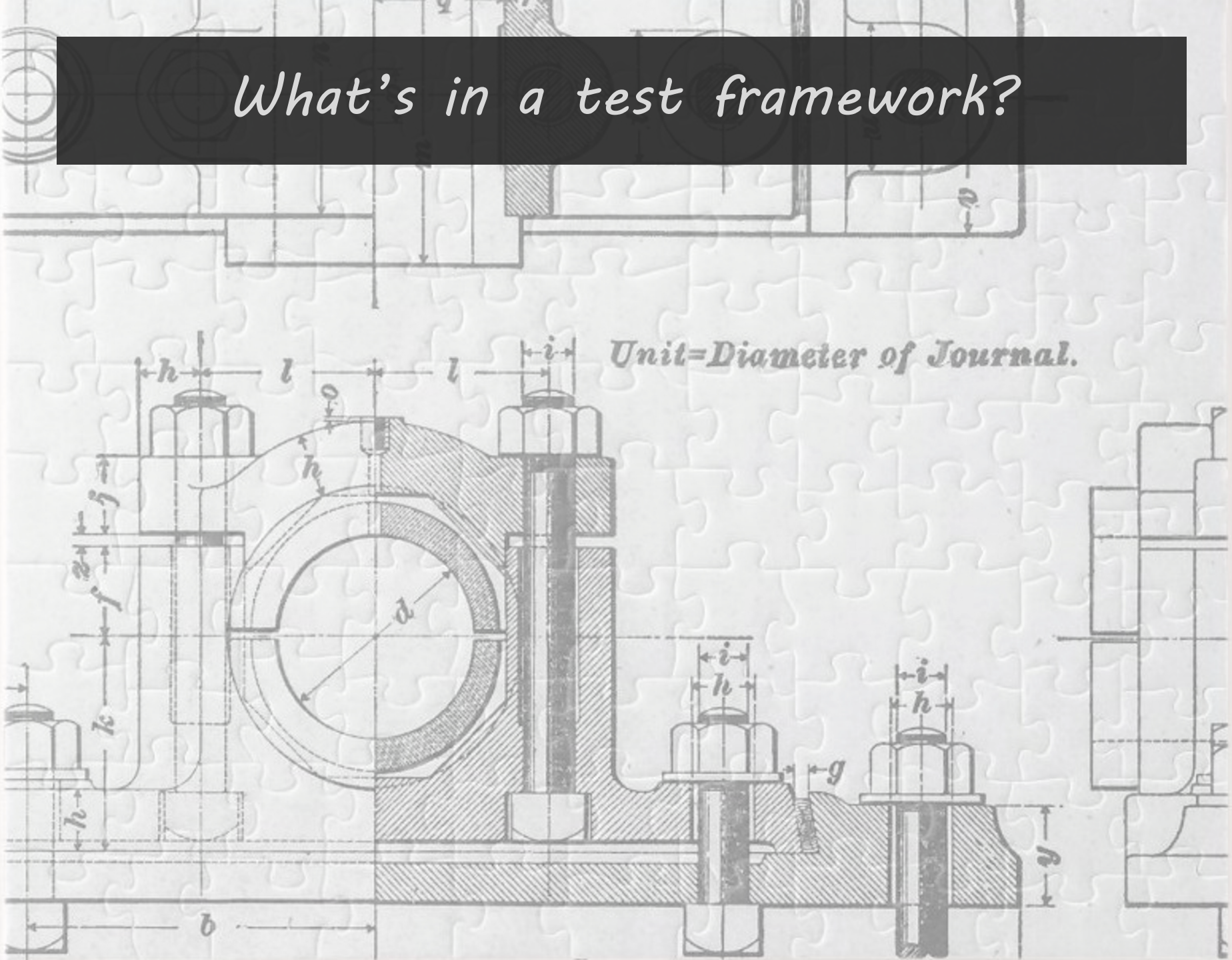
TestRail

MICRO FOCUS ALM Quality Center

Cloud Platforms

Continuous Integration & Delivery

What's in a test framework?



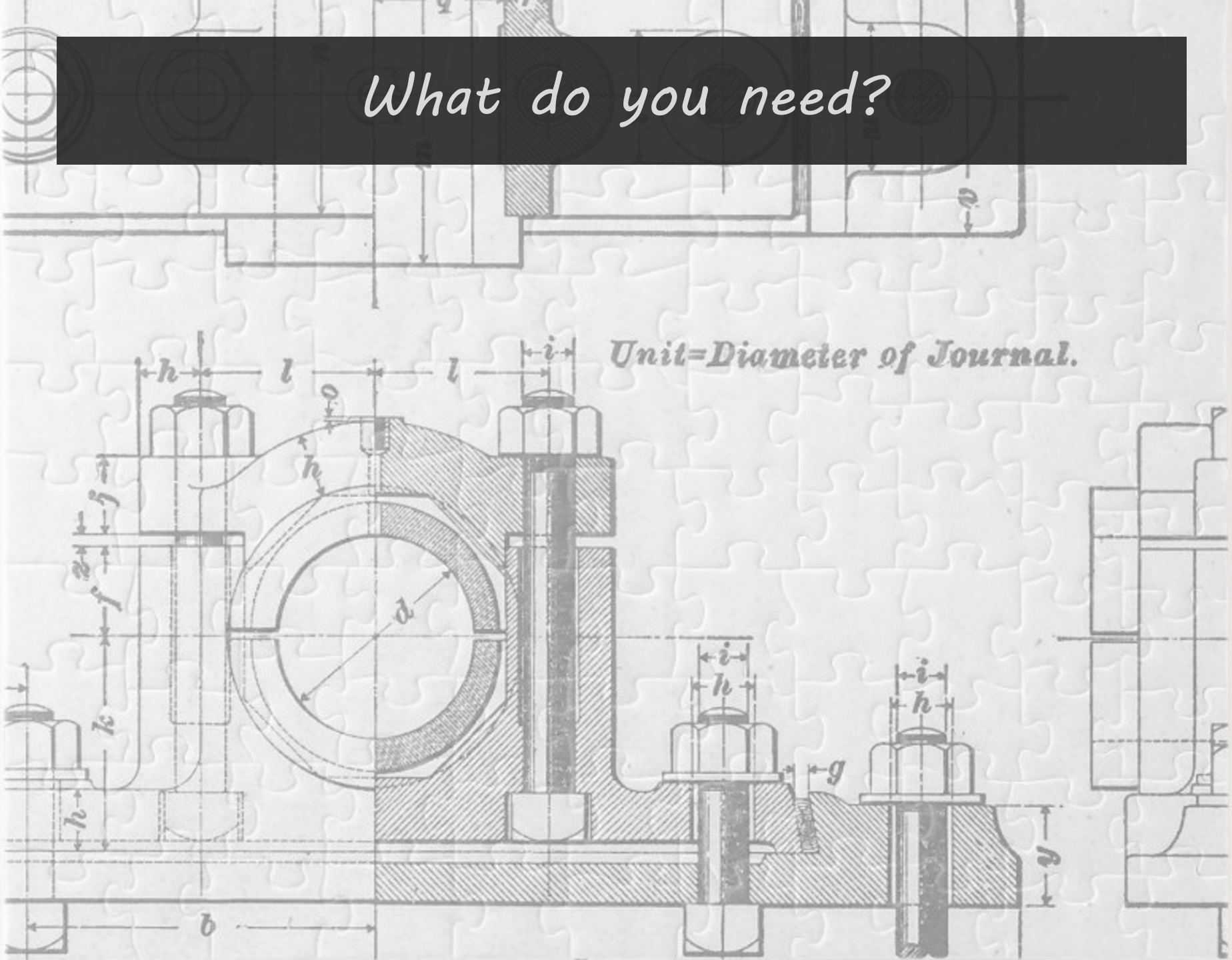
What's in a test framework?

- Tools
- Libraries
- Test Architecture
- Reusable Automation Code
- Standards
- Test Development Process
- Reporting
- Analysis & Strategy
- Maintainability
- Reusability
- Feedback cycle

What's NOT in a test framework?

- Tests
- Requirements
- Test Environments
- Test Data
- Development Tools
- Collaboration Process
- Development Workflow
- Project Management
- Infrastructure

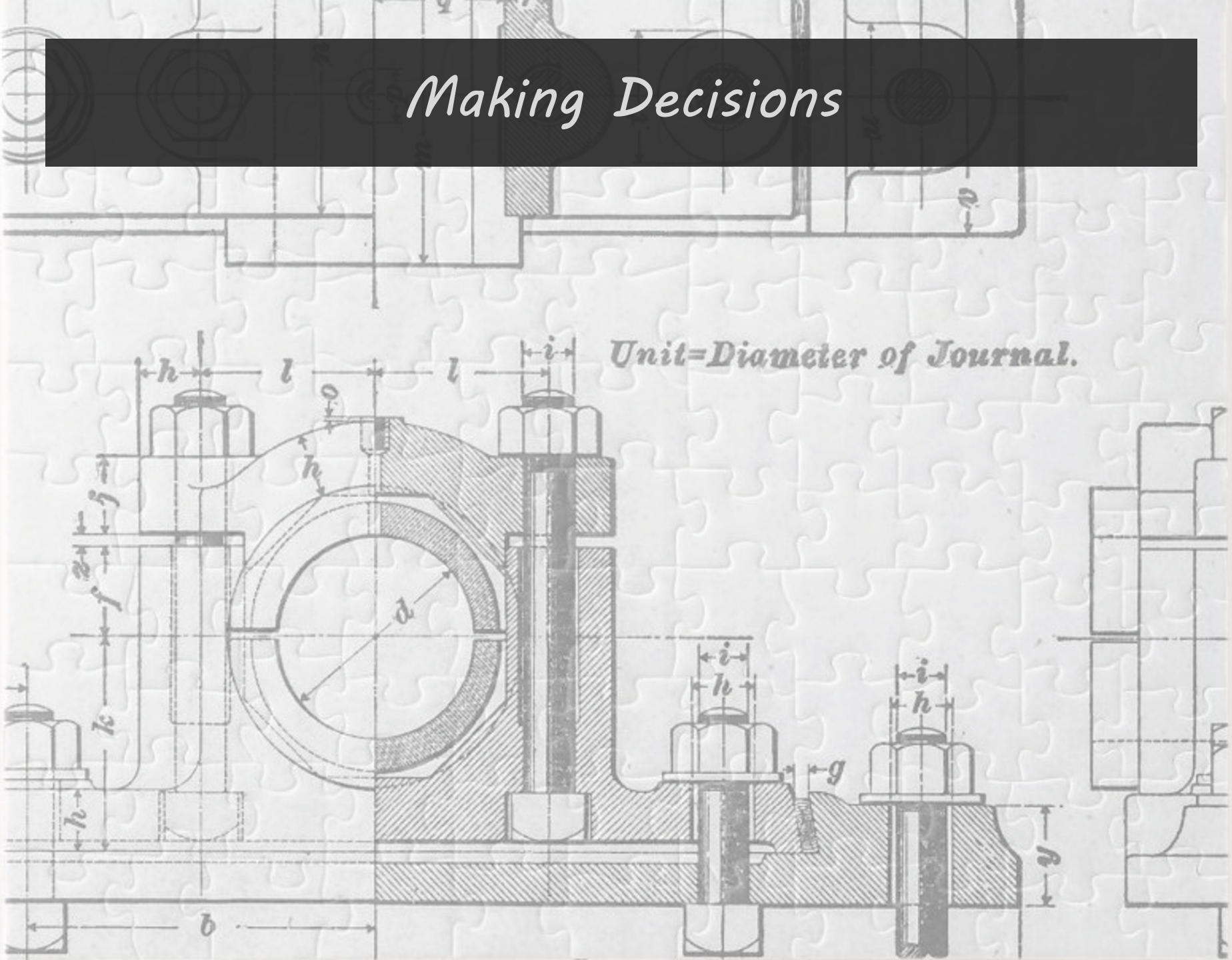
What do you need?



What do you need?

- Fast test execution and feedback
- Reliable and repeatable test results
- Easy to write and maintain test automation
- Clear reporting and analytics
- Test case management & requirements coverage
- Collaboration with developers, manual testers, product owners
- Continuous delivery and execution on demand
- Non-functional: Load, performance, accessibility, security tests
- GUI, API, system, integration, unit tests
- Targeted and full regression test groups
- Test against all environments: DEV, QA, STAGE, PRODUCTION

Making Decisions



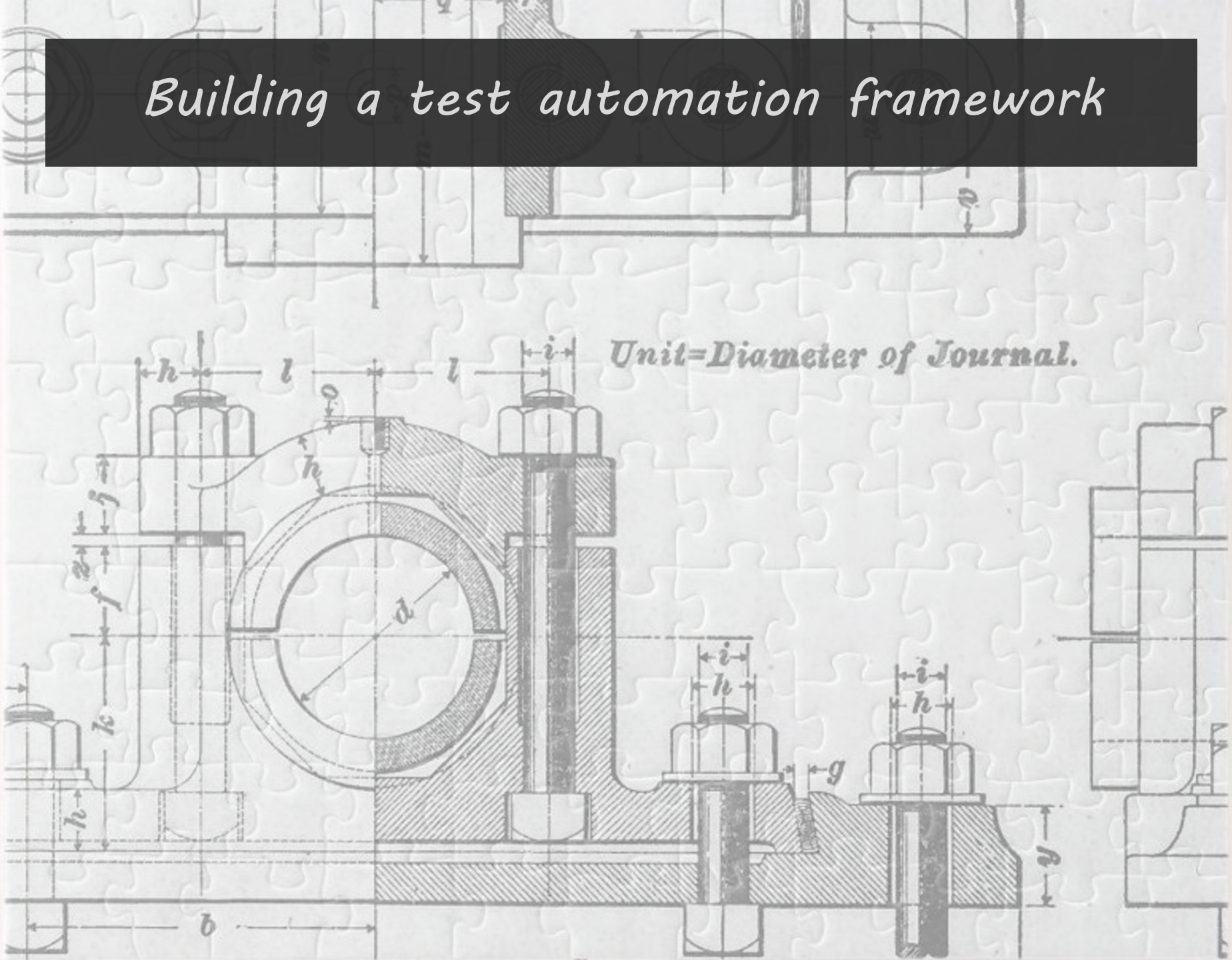
What do you need to do?

- Develop test automation
- Version control
- Resuable code
- Execute tests
- Trigger when to execute tests
- Decide which tests to execute at any given time
- Keep track of results
- Analyze results over time
- Configuration for different environments and platforms
- Generate or retrieve test data
- Debug and fix broken tests

What do you need to do?

- Is my software working properly?
- Which tests executed?
- Which requirements are covered by tests?
- What systems were tested in which environment?
- How long it took to execute?
- What caused any false failures?
- What issues escaped into production?

Building a test automation framework



Building a test automation framework

Getting Started
Test driven test development
Putting the pieces together
Test automation principles
Iterative development
Continuous feedback loop
Build for scalability
Keeping it green
Refactoring

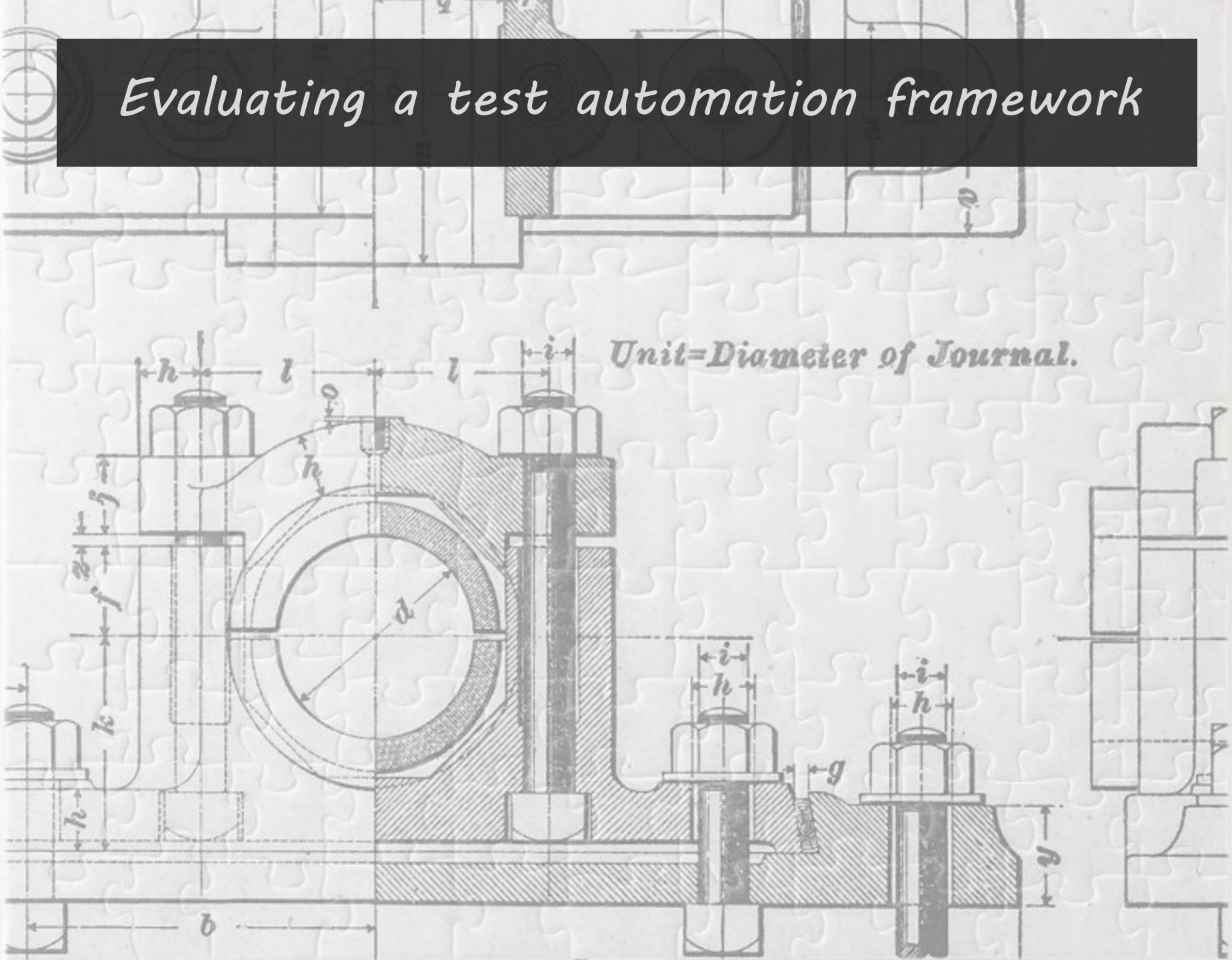
Test Automation Principles

- KISS (Keep it simple and stupid)
- DRY (Don't repeat yourself)
- SOLID (especially single responsibility principle)
- Separation of concerns (separate tests from implementation details)
- Consistency (patterns and code structure)
- Reliability & Repeatability
- Run anywhere (no more "It works for me")
- Atomic and autonomous tests (avoid interdependency)
- Test at different levels (not just end-to-end)
- Data driven test scenarios
- Reusable steps, setup, validation, etc.
- Version control and traceability

Pieces of the Puzzle

- Programming language
- Dependency management
- Project structure
- Configuration and environment settings
- Logging and Debugging
- Security & Secrets Management (passwords, API keys, etc)
- Test management and requirements
- Code structure and patterns
- Test execution and filtering
- Reporting and notification
- Remote execution and parallelization
- Target platforms: Browsers, Mobile Devices, Operating Systems

Evaluating a test automation framework



Evaluating a Test Automation Framework

- What are your goals / needs?
- What is your experience?
- What are you missing?
- What do you want to learn?
- What do you want to improve?
- Who is the target?

Evaluation Points

- Principles being followed (Kiss/DRY/OOP, etc)
- Architecture
- Scalability
- Maintainability
- Reusability
- Time to identify / fix an issue
- Time to develop new automation
- Execution speed
- Identify requirements coverage
- Ability to run a specific subset
- Security, secrets

Acting on an evaluation

- Most important issues
- Biggest impact
- Training and hiring
- Refactor or rewrite
- Changing the wings in flight
- Timeline
- Budgeting for improvement

Measuring Success

- Define success criteria
- Rank by importance / impact
- Failure rates
- Test execution speed
- Speed to fix issues
- Accuracy identifying issues
- Ability to write new tests
- Time to have new testers become effective