CodeLogic

Case Study:
Application Re-Engineering

Highlight

After throwing out a whole codebase, this team **decreased triage time by 53% and established a continuous flow** allowing them to move fast and deliver predictably while ensuring debilitating technical debt will never swallow up productivity — or their codebase — ever again.

Continuous Scanning

Application Dashboard

IDE Integration

Overview

A dev team was ready to wrap up a rescue project for a failing application when they arrived at the painful conclusion that it needed a software rewrite. **Team leadership had to reconcile two contradictory needs: moving fast while simultaneously keeping things clean.** For speed, they needed to empower several dev teams to make strategic software development and architecture decisions quickly. At the same time, they needed to keep the software architecture solid to avoid tanking this critical application again.

Approach

To start, the team chose to implement the CodeLogic Software Intelligence Platform. The lead architect regularly reviewed CodeLogic's dependency mapping and complexity insights. If the data access layer showed inappropriate coupling across the system, he'd work with the team to keep the code clean. As methods got long or overly complex, the team would refactor. When similar integrations diverged on which API endpoints they used, the team leads were brought together briefly to decide on the right path and move forward with more clarity and confidence. *(continued)*

"Before CodeLogic, when we'd overhaul a shared library, it would always be a mess. We'd miss things that would only hit us when building other modules, or even in testing. It was difficult to know when we were truly done. Now it's so much easier to see the 'blast radius' of a change and plan accordingly"

Over the course of a few months, the frequency of interventions decreased. The developers learned that taking on technical debt to get a task done a little quicker resulted in extra work later, when the debt was discovered. While occasionally taking on tech debt is the right decision, the team learned to consciously make that decision together rather than individually.

Results

Unlike architects who draw diagrams and hope for the best, this team's lead architect proactively used the CodeLogic Application Dashboard to identify bad patterns in the software architecture as they emerged in order to address them quickly.

"Some of this we should have caught in code review, but many of our issues looked fine commit to commit. Only when you're able to see the full picture across libraries and services do these issues become obvious." - Lead Architect

Highlight

17% increase in dev team productivity

53% decrease in weekly triage mtgs

Using CodeLogic when planning bigger changes also helped the team execute those changes better. "Before CodeLogic, when we'd overhaul a shared library, it would always be a mess. We'd miss things that would only hit us when building other modules, or even in testing. It was difficult to know when we were truly done. Now it's so much easier to see the 'blast radius' of a change and plan accordingly" observed a lead developer.

Too often, smaller changes to a shared library cause issues. Although checking CodeLogic before making changes would prevent those issues, developing a new habit of checking the platform was difficult for many team members. As a result, the team decided to add CodeLogic's integration to their IDE: JetBrains IntelliJ. Now, when the team checked references in the IDE as part of their existing workflow, CodeLogic provided additional insights within the IDE. When an interesting reference surfaced, they could slide into the CodeLogic WebUI to learn more.

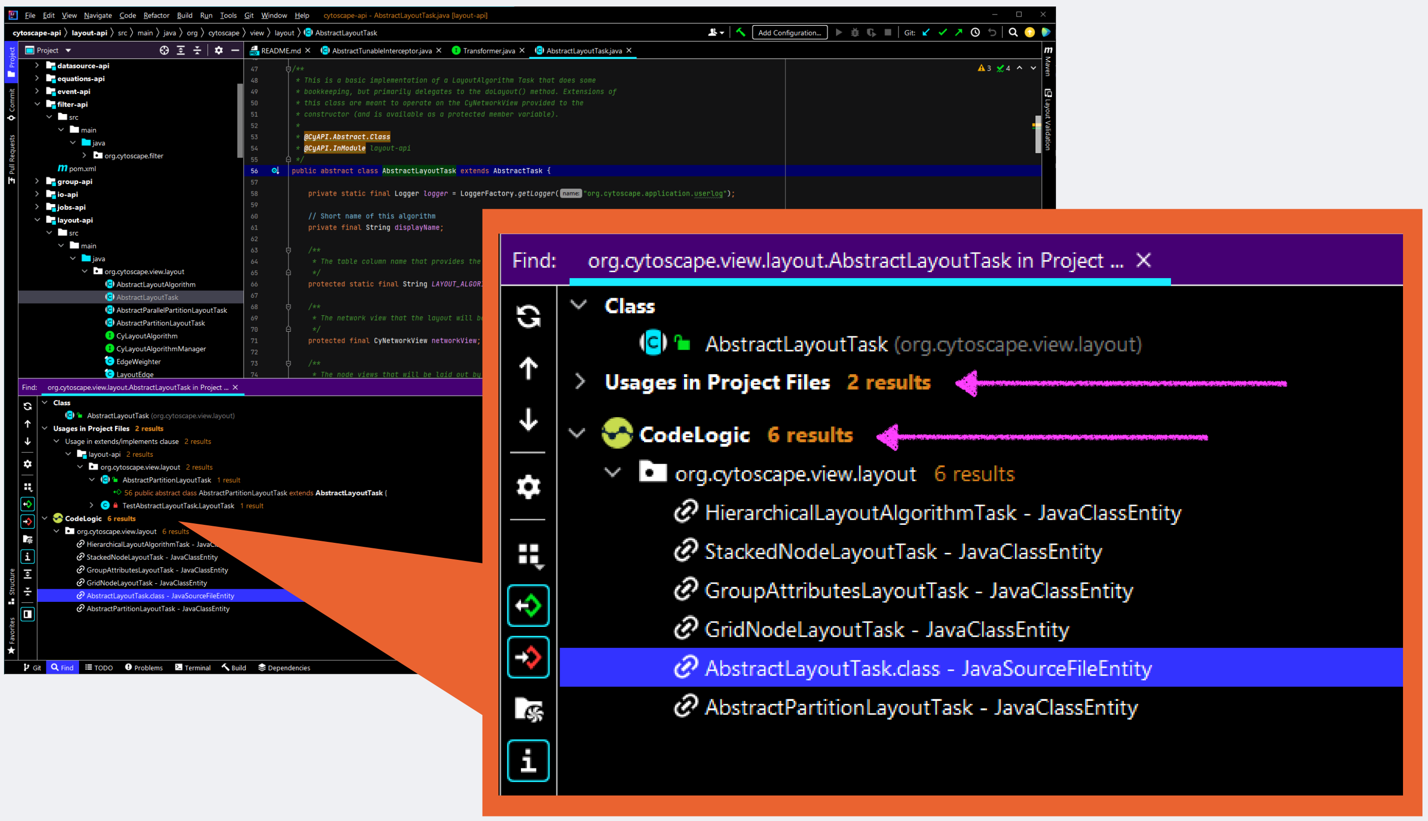
As one developer put it, "Accessing CodeLogic within IntelliJ means we don't have to break our focus to find the information we need – We're already there."

Conclusion

Now, with CodeLogic integrated into their continuous workflow, the source of code defects dried up, saving significant development and testing time, and shortening weekly triage meetings by 53 percent. Because those errors often impacted people on other teams, reducing them helped build confidence across teams and keep the larger organization harmonious and happy.

For the executive overseeing the team, the CodeLogic benefits are clear, **“Day-in and day-out, we’re saving 17% of the engineering team’s time. But the real benefit is that we’re keeping the code clean enough to go fast, deliver predictably and never throw out a whole codebase again.”**

Today, this organization is still moving quickly. Their foundations of good architecture are well maintained. They continue to avoid creating creeping technical debt that swallows productivity and leaves everyone miserable. Now, the teams are empowered to make good decisions, with the right information at their fingertips. Thanks to CodeLogic’s features and integrations, they are also better-equipped to know when to refactor, and when the software needs a rewrite, much earlier in the game.


cytoscape-api - AbstractLayoutTask.java [layout-api]
README.md
AbstractTunableInterceptor.java
Transformer.java
AbstractLayoutTask.java
Add Configuration...

```
/**
 * This is a basic implementation of a LayoutAlgorithm Task that does some
 * bookkeeping, but primarily delegates to the doLayout() method. Extensions of
 * this class are meant to operate on the CyNetworkView provided to the
 * constructor (and is available as a protected member variable).
 *
 * @CyAPI.Abstract.Class
 * @CyAPI.InModule layout-api
 */
public abstract class AbstractLayoutTask extends AbstractTask {

    private static final Logger logger = LoggerFactory.getLogger( name: "org.cytoscape.application.userlog");

    // Short name of this algorithm
    private final String displayName;

    /**
     * The table column name that provides the
     */
    protected static final String LAYOUT_ALGOR

    /**
     * The network view that the layout will b
     */
    protected final CyNetworkView networkView;

    /**
     * The node views that will be laid out by
```

Find: org.cytoscape.view.layout.AbstractLayoutTask in Project ...

```
Class
    AbstractLayoutTask (org.cytoscape.view.layout)
Usages in Project Files  2 results
CodeLogic  6 results
    org.cytoscape.view.layout  6 results
        HierarchicalLayoutAlgorithmTask - JavaClassEntity
        StackedNodeLayoutTask - JavaClassEntity
        GroupAttributesLayoutTask - JavaClassEntity
        GridNodeLayoutTask - JavaClassEntity
        AbstractLayoutTask.class - JavaSourceFileEntity
        AbstractPartitionLayoutTask - JavaClassEntity
```

CodeLogic IDE integrations identify references that native IDEs do not detect.