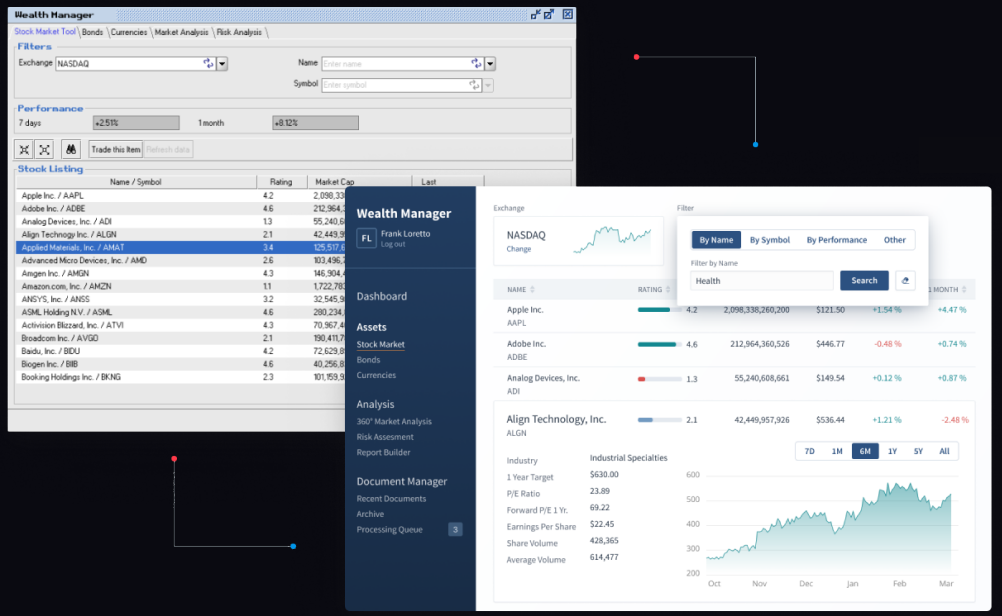


Modernizing Java Swing Applications

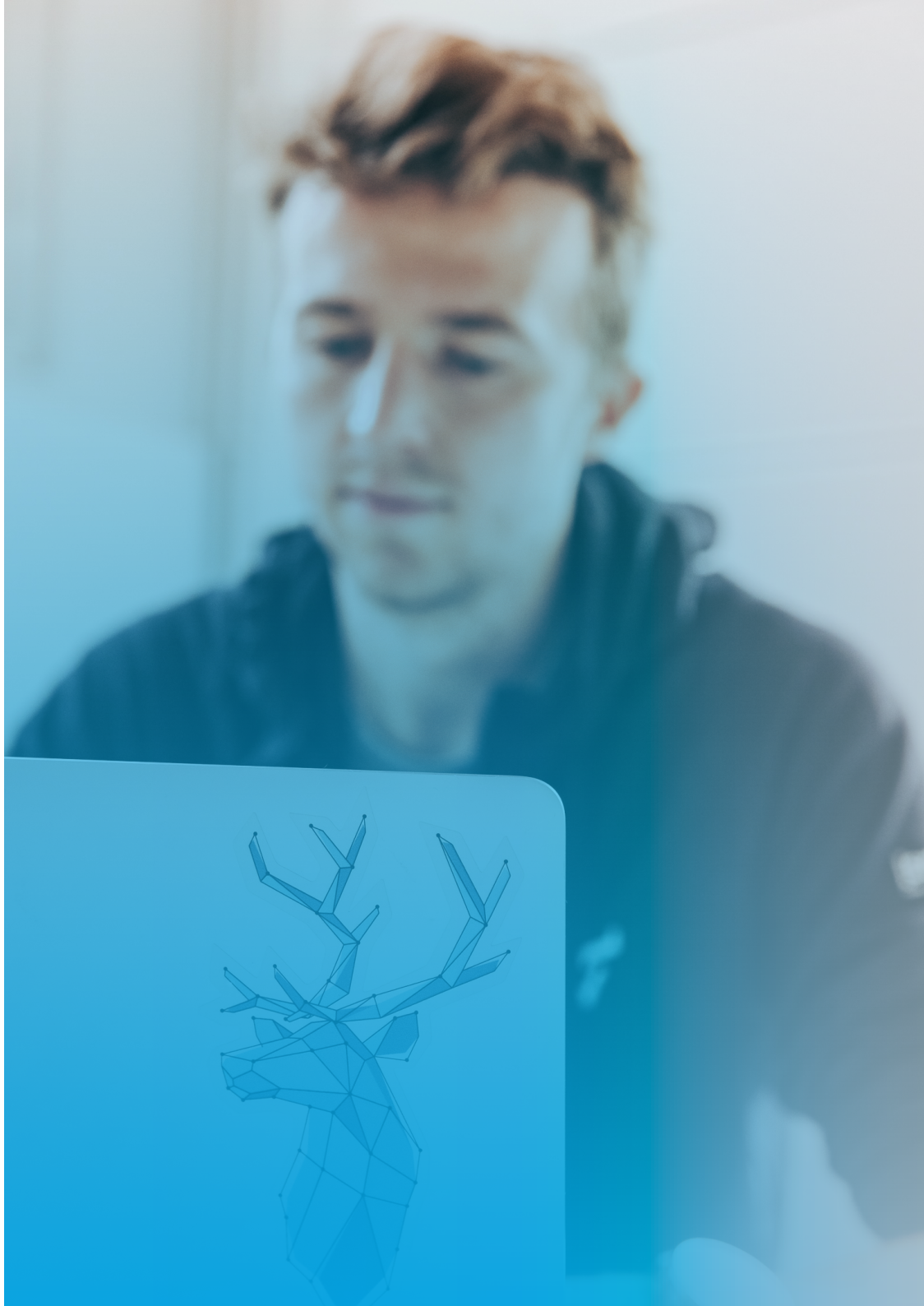
A Five-Step Blueprint for Decision-Makers

vaadin}>

Table of contents

Executive summary

Modernizing a Java Swing application for the web can eliminate legacy dependencies, reduce the maintenance burden, ensure compatibility with modern web technologies, and enhance the user experience, among other advantages. Many traditional Java modernization projects, adopting a phased approach, can take years, with developers often lacking the motivation to finish. The automation provided by the Vaadin Modernization Toolkit can refactor the bulk of your Java code, enabling you to kill the tail on legacy dependencies quickly. This offers the fastest route to migrate your working Java applications away from outdated frameworks.



Introduction

Migrating a Java Swing to a web application – or any legacy application, for that matter – can be a complex and time-consuming process. Despite the challenges, however, the benefits of application modernization far outweigh the difficulties. And with the right tools, a Swing migration can be a relatively seamless process.

Whether you are looking to improve system performance, meet current security standards, or simply provide a better user experience, the insights offered in this white paper will guide you. It details the benefits of updating your software environment and explains how Vaadin can simplify and speed up your transition from Swing to Web with automated tooling.

The Clock is Ticking for Swing Users

You might be questioning the urgency behind modernizing Swing applications, especially when Oracle still supports Swing and your applications are functioning without issues. However, waiting too long to migrate your mission-critical applications away from Swing could be as harmful as rushing to be the first to do so.

Staying with Swing presents multiple challenges that affect not just your application but also your end users, development team, and the overall competitiveness of your company. Managing client installations becomes notably more cumbersome on Swing, and its applications are incompatible with certain devices, such as iPads. Furthermore, in an era where web alternatives are preferred, desktop applications, including those built on Swing, are perceived as outdated.

Moreover, time is not on your side. The cost of migration is unlikely to decrease over time. This is primarily because technological advancements and inflationary pressures tend to increase the expenses associated with software development and migration efforts over time. Additionally, Swing has not been included in college curricula for many years, resulting in a scarcity of new developers equipped with the necessary skills to understand or modernize your application.

Lastly, consider the aspect of innovation. The ecosystem for Swing tools has not evolved for quite some time, making it challenging for your developers and organization to leverage the extensive Java open source work available. Even if your Swing application is performing satisfactorily within your organization without immediate security concerns, its reliance on Swing is likely hindering your innovation capabilities and ability to stay competitive in the fast-evolving technological landscape.

Moving Beyond Java Swing: The Benefits of Modernization

The business benefits of migration typically fall into two categories: removing the constraints imposed by the old technology and unlocking the opportunities introduced by the new technology.

Eliminating constraints

Ever-Increasing Maintenance Costs

All software systems accumulate technical debt, essentially a “tax” that has to be paid to address issues that arise from using legacy technology. According to McKinsey, companies often find themselves paying an additional 10 to 20% on top of project costs to manage and mitigate this tech debt.

The old code forces your development team to spend more and more time maintaining it, which detracts from the time available to implement new features. Working with old technology also becomes more time-consuming as the team tries to fulfill new, non-functional requirements, such as performance or scalability, that would be simple to implement with the new technology. Reducing technical debt enables engineers to dedicate up to 50% more of their time to projects that actually generate value.

Slow and Costly Deployment

Unlike web applications, delivering and keeping desktop applications synchronized with the latest updates is slower and more error-prone. In addition, the setup and maintenance costs of the infrastructure required to deliver and deploy them are significantly higher than for web applications.

Losing Market Share to the Mobile Web

The shift from desktop to mobile caused the end of the desktop-only era, leading to a significant loss in market share for platforms unable to adapt. In 1999, a Swing application could work on almost any graphics-enabled computing device without recompiling, yet less than half of the devices sold today can run native Swing.

This lack of innovation hinders user experience and can lead to revenue losses. The shift is not just in consumer behavior; mobile devices have also become prevalent in business environments, necessitating applications that are accessible across tablets, phones, and desktop computers. The inability to support mobile platforms has resulted in a considerable reduction in market share, underscoring the importance of adapting to the mobile web.

Security Vulnerabilities

Legacy systems can pose a significant cybersecurity risk due to accumulated vulnerabilities and inadequate patching practices over the years. Modernizing these systems by enhancing security measures and ensuring compliance with industry standards and regulations is essential. Such modernization efforts are crucial for reducing the likelihood of security breaches and avoiding the financial repercussions of non-compliance.

For instance, strict privacy laws, including the General Data Protection Regulation (GDPR), California Consumer Privacy Act (CCPA), and Health Insurance Portability and Accountability Act (HIPAA), can impose significant fines on organizations that fail to safeguard their customers' data adequately. According to research conducted by IBM, the global average cost of a data breach in 2023 climbed to $4.45 million, representing a 15% increase over the last three years, highlighting the financial stakes of ensuring robust cybersecurity and compliance measures.

Unlocking New Opportunities

Vaadin brings you a set of award-winning web components that can ensure a smooth transition from Swing to the modern web application your users expect. Choosing Vaadin to web-enable your Swing applications offers many benefits:

Developer Productivity

Vaadin is the only well-supported UI framework for the Web with a Swing-like programming model. Vaadin applications are built using Java and support all the patterns your developers use to keep code organized. The platform abstracts away the details of browser execution with a Java API that keeps developers productive and makes them feel right at home with the new stack.

Modern Components

With Vaadin, you will get access to a regularly updated set of state-of-the-art components that pass the most stringent accessibility compliance tests, are coherently themable, work consistently on all devices, and have great UX.

Modern Java and Modern Web

Finding new developers to maintain your Vaadin application has never been easier. Vaadin uses APIs with the newest Java versions like Streams, Optionals, and Lambdas. Vaadin has also been completely redeveloped with the latest browser standards, so with basic web and Java skills, new developers will be productive quickly. In addition, feature updates are instantly available on the web for all users, removing the need to upgrade desktop software.

Enhanced Security and Compliance

Vaadin Flow is a server-side framework where all of your application's state, business, and UI logic resides on the server. Unlike client-driven frameworks, a Vaadin application never exposes its internals to the browser, where vulnerabilities can be leveraged by an attacker. This makes it inherently more secure than many of the alternatives.

A Cost-Effective Transition

Vaadin offers one of the most cost-effective paths to web-enable your valuable applications - more code can be reused, and what can't be reused can often be refactored. By leveraging more of the code that works today, there will be less need for testing and troubleshooting due to fewer defects.

Support and Resources

Vaadin supports you both during your modernization project and with tools and resources afterward. During your project, Vaadin offers software tools for understanding and refactoring Swing applications. Once your project is finished, Vaadin support is available to help bridge any hypercare period and beyond to provide fixes for new problems introduced by your browser updates and the ever-changing world of security.

How Vaadin Simplifies Modernization from Swing to Web

The Vaadin Flow framework is the most widely adopted full-stack UI framework for writing modern web applications entirely in Java. Vaadin allows developers to continue developing their applications in 100% Java, making the transition from Swing as smooth as possible.

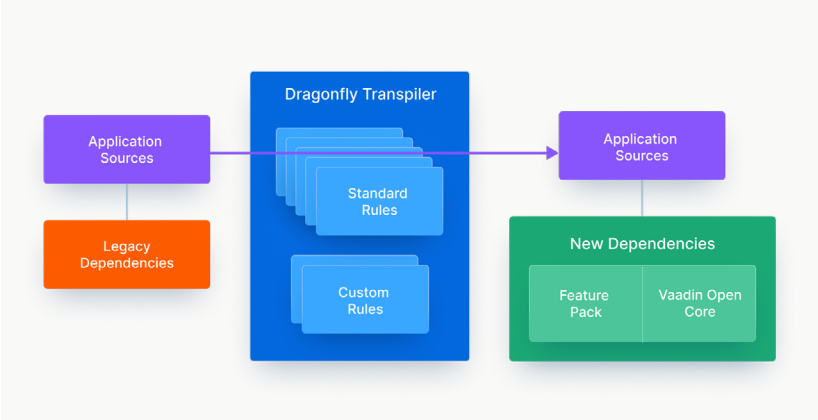
Vaadin Flow's unique architecture simplifies Java application modernization, reducing effort by integrating the layers and technologies needed for UI and communication into one, unlike frameworks such as Angular.

We also provide up to 15 years of support for a Vaadin version, so you won't have to think about migrating to a new technology again any time soon!

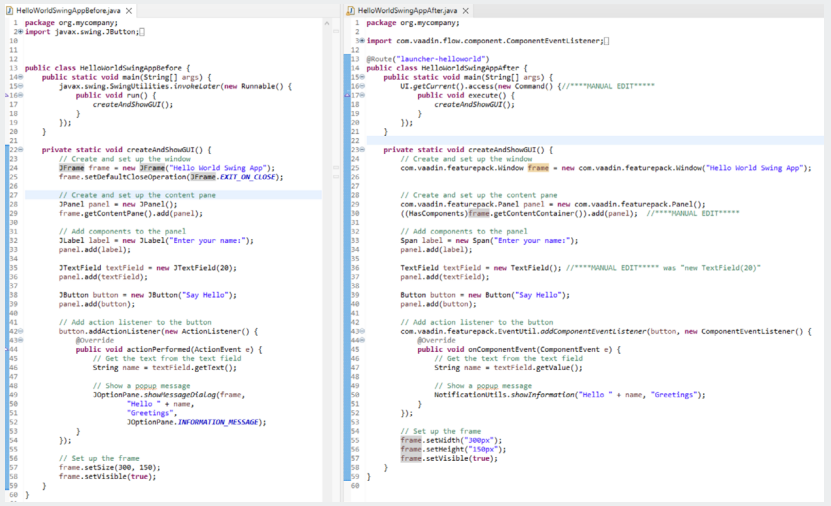
```
import com.yourapp.backend.GreetService;

public HelloWorldView(GreetService service) {
    TextField name = new TextField("Name");
    Paragraph greeting = new Paragraph("");
    Button sayHello = new Button("Greet");

    sayHello.addClickListener(e -> {
        greeting.setText(
            service.greet(name.getValue()));
    });
}
```

Simplified example: Creating web UI that interacts with Java (Spring) backend service. Vaadin Flow facilitates a secure connection between UI and the backend service.

Automating your Swing to Web Migration



The Vaadin Modernization Toolkit offers a streamlined solution for transitioning from Swing to Vaadin, simplifying the process of converting Swing applications into modern web applications. Our Java-to-Java transpiler significantly reduces the investment required for such a transition. The Toolkit includes a source code refactoring tool, feature packs, and an option for phased migration, allowing for flexibility in how the migration is approached. The refactoring tool can be utilized once or multiple times throughout a project, adapting as the rules it applies are refined.



HelloWorldSwingAppBefore.java

```
package org.mycompany;
import javax.swing.JButton;


public class HelloWorldSwingAppBefore {
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }

    private static void createAndShowGUI() {
        // Create and set up the window
        JFrame frame = new JFrame("Hello World Swing App");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create and set up the content pane
        JPanel panel = new JPanel();
        frame.getContentPane().add(panel);

        // Add components to the panel
        JLabel label = new JLabel("Enter your name:");
        panel.add(label);

        JTextField textField = new JTextField(20);
        panel.add(textField);

        JButton button = new JButton("Say Hello");
        panel.add(button);

        // Add action listener to the button
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Get the text from the text field
                String name = textField.getText();

                // Show a popup message
                JOptionPane.showMessageDialog(frame,
                        "Hello " + name,
                        "Greetings",
                        JOptionPane.INFORMATION_MESSAGE);
            }
        });

        // Set up the frame
        frame.setSize(300, 150);
        frame.setVisible(true);
    }
}
```

HelloWorldSwingAppAfter.java

```
package org.mycompany;

import com.vaadin.flow.component.ComponentEventListener;

@Route("launcher-helloworld")
public class HelloWorldSwingAppAfter {
    public static void main(String[] args) {
        UI.getCurrent().access(new Command() {//****MANUAL EDIT*****
            public void execute() {
                createAndShowGUI();
            }
        });
    }

    private static void createAndShowGUI() {
        // Create and set up the window
        com.vaadin.featurepack.Window frame = new com.vaadin.featurepack.Window("Hello World Swing App");


        // Create and set up the content pane
        com.vaadin.featurepack.Panel panel = new com.vaadin.featurepack.Panel();
        ((HasComponents)frame.getContentContainer()).add(panel);  //****MANUAL EDIT*****

        // Add components to the panel
        Span label = new Span("Enter your name:");
        panel.add(label);

        TextField textField = new TextField(); //****MANUAL EDIT***** was "new TextField(20)"
        panel.add(textField);

        Button button = new Button("Say Hello");
        panel.add(button);

        // Add action listener to the button
        com.vaadin.featurepack.EventUtil.addComponentEventListener(button, new ComponentEventListener() {
            @Override
            public void onComponentEvent(ComponentEvent e) {
                // Get the text from the text field
                String name = textField.getValue();

                // Show a popup message
                NotificationUtils.showInformation("Hello " + name, "Greetings");
            }
        });

        // Set up the frame
        frame.setWidth("300px");
        frame.setHeight("150px");
        frame.setVisible(true);
    }
}

```

This simplified example demonstrates how the transpiler preserves the code structure, though some lines may still require manual work.

Designed to automate the migration of a portion of your application's source code, the Toolkit's effectiveness is further boosted by our team's ability to manually fine-tune the migration, ensuring it precisely meets the specific needs of your application. This combination of automation and customization makes the Vaadin Modernization Toolkit an invaluable resource for modernizing legacy applications.

Fine-Tuning

Application modernization projects usually come with significant risks, mainly because teams often embark on such projects for the first time and lack the experience and guidance needed for effective planning. With Fine-tuning, teams can leverage the expertise of Vaadin experts who bring practical experience and share the burden of risk.

The Fine-tuning solution transforms your application to a predefined level of automation, ensuring each line of code is accurate and reliable, all at a fixed cost. All applications are unique, and any two applications can rely on the same framework but different versions or APIs, so it's to be expected that Fine-tuning would be useful with any application.

Feature Pack

As a platform for building modern web applications, Vaadin aligns with the evolving best practices around "mobile-first" and "mobile-also" development. However, many older applications were optimized for "desktop-only" use, mainly serving knowledge workers. The developers of these applications have used a number of features of their frameworks to deliver value to users efficiently.

The feature packs of the Vaadin Modernization Toolkit reintroduce these capabilities. While these features are typically associated with desktop environments in practice, they can be applied to any component of an older framework. For example, consider how Swing divides the responsibilities for containing and arranging child components between Panels and LayoutManagers. The feature packs support these in addition to the arrangement strategy of GridLayout, BorderLayout, FlowLayout, and others.

A Five-Step Path to a Successful Swing Migration

The Vaadin Modernization Team has successfully assessed over 80 large-scale Java applications. Utilizing our award-winning tools and expertise, we conduct a comprehensive assessment of technical risks and devise tailored strategies for modernizing your application.

Various strategies for migrating a Swing application exist, yet most successful projects share several common phases. In this section, we'll outline these phases across five steps.

However, before taking steps in any direction, it makes sense to ensure that the direction is right. Proof of concepts are an ideal way to test the waters of a new stack and ensure it's complying with the full range of non-functional requirements your team and users have moving forward. Proof of concepts take many forms and can have a range of scopes.

One obvious consideration involves developers questioning whether they are comfortable with the responsibility of maintaining and supporting this application if it runs on "x" technology. It's also important to assess if developers are capable of creating a product that existing users will ultimately want to switch to. In addition, maintainability and usability are issues frequently addressed in proof of concepts because they directly impact the application's principal stakeholders.

How far should it go? That depends entirely on the application and the way it is used. If the application's scalability is a big concern, scalability should also be part of the proof of concept, likewise, for performance, availability, portability, accessibility, security, or any of the other non-functional attributes.

Now, let's explore the key steps involved in achieving a successful Swing migration.

Step 1: Migration Assessment and Tech Review

With a successfully completed proof of concept behind them, stakeholders will be justified in looking at the migration more closely as a project. This is where a formal assessment will bear fruit, and you can look into quantifying and qualifying how the application depends on Swing, AWT, the desktop, and all the peripherals it's connected to. Technical experts at Vaadin can perform a variety of assessments on Swing applications ranging from full-blown Migration Assessments to shorter assessments geared to understanding the impact of migrating with a specific conversion tool.

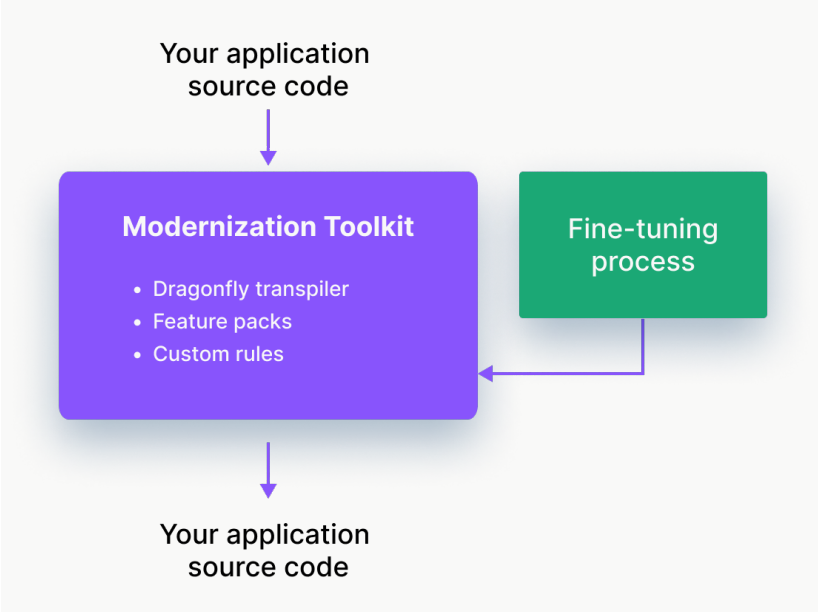
"Assess" is one of the three pillars of Vaadin's tooling for Swing. Vaadin's assessment tools work based on a formal parsing of your Java sources and your application's build to deeply resolve all types it uses. This way, we can determine if a method invocation like pack() is related to the method in class `javax.swing.JInternalFrame` or some other method declared in your application that just happens to have the same name. Having identified the Swing and AWT references, we now have a clearer understanding of the changes needed in your application to transition it to Vaadin.

This data also reveals the extent to which Vaadin's automated refactoring tool accommodates the specific Swing APIs and classes your application utilizes, preparing us for the next step: fine-tuning the Modernization Toolkit.

Step 2: Fine-Tuning the Conversion Model

All applications are unique, and it's normal that any automated refactoring tools and Feature Packs need to be adjusted to consider this uniqueness. Technical experts at Vaadin can fine-tune the Modernization Toolkit to improve its coverage of the APIs and customizations your application uses.

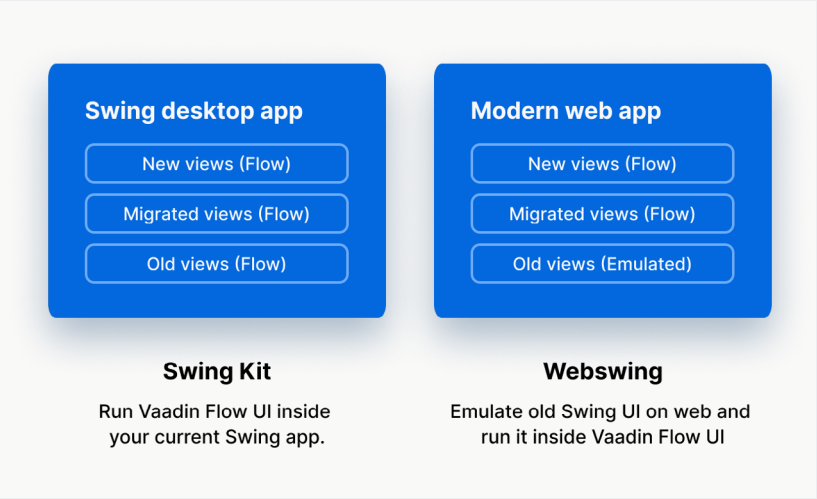
To facilitate the process for stakeholders without experience with semi-automated application modernization, Vaadin can offer the fine-tuning project to an agreed coverage percentage at a fixed price, complete with a guarantee of successful results.



Step 3: Legacy Runtime Powers Incremental Migration

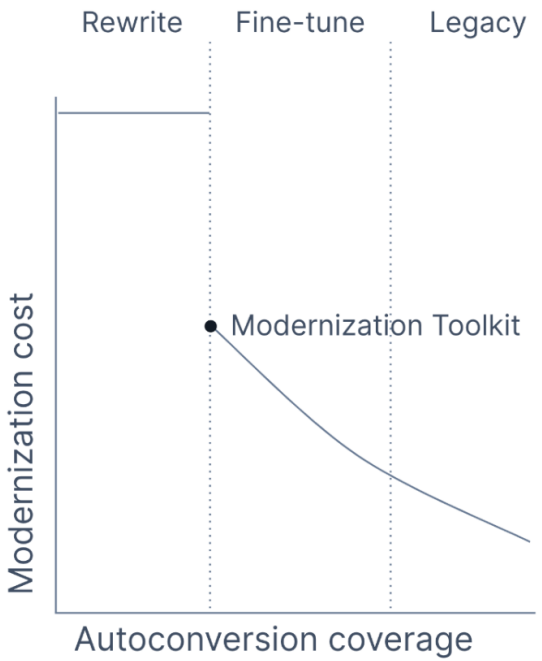
The actual migration phase can be either phased or big bang. The phased migration approach means you would gradually migrate your application incrementally to the new tech stack while keeping the old and new parts runnable side by side. For this purpose, Vaadin has a Swing Kit to run Vaadin-based views inside a Swing application. An alternative is also to use Webswing with its migration modules specifically geared to Vaadin compatibility.

Although a phased approach benefits users and testers, a big-bang migration is typically the most cost-efficient method. In most cases, migration from Swing to Vaadin can be done semi-automated using Vaadin's migration tooling. Tool-assisted migration is an iterative process where each iteration involves adjusting the automation tools to suit the specifics of the application being migrated, running these tools, and then analyzing the results through testing. After achieving the optimal level of automation, the remaining migration tasks can be completed manually.



Step 4: Manual Migration

After utilizing the Vaadin Modernization Toolkit to migrate the bulk of the Java code to modern Vaadin Flow code, the remaining parts must either be manually converted, left to operate on a legacy runtime, or completely redesigned and rewritten. This conversion process can be undertaken incrementally, handling the application view by view over a transition period. This approach ensures continued development and allows the application to remain in production throughout the migration process.



Step 5: Re-Design the Chosen Views

The final step involves the strategic selection and redesign of the application's most impactful views, a phase where the full potential of modernization is realized. By focusing on these key areas, the process ensures the application meets modern standards and capitalizes on new opportunities for improvement. Vaadin Flow provides a productive and stable foundation for both modernization and future development efforts.

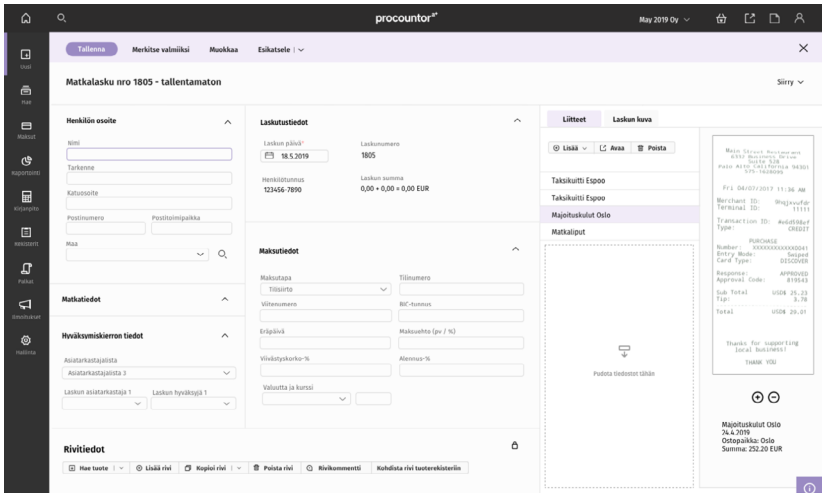
Simply transitioning away from outdated technologies can bring immediate benefits, such as making your application compatible with modern web browsers and devices, enhancing accessibility, improving the overall look and feel, boosting performance, and facilitating the application's move to cloud computing. This comprehensive redesign step can significantly improve your application's user experience and operational efficiency.

How HPD Lendscape and Procountor Succeeded with Their Migrations to Vaadin

Don’t just take our word for it. Explore the real-life case studies below to see how other companies have successfully migrated from Swing to Vaadin.

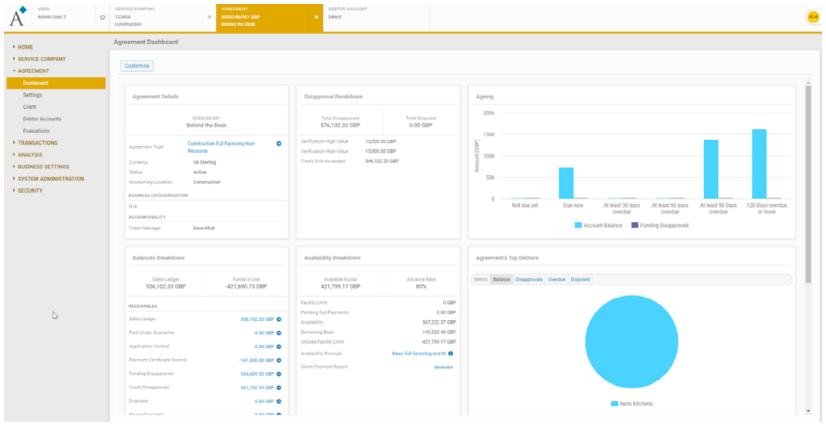
Procoutor Oy

Procountor Oy successfully migrated its accounting software from desktop to the web using Vaadin. In just two years, they efficiently replaced Swing implementations with Vaadin, emphasizing a user experience (UX)-first approach. This migration allowed Procountor Oy to modernize its software, making it more accessible and user-friendly by moving it to a web-based platform. Vaadin's framework was key in this seamless transition, helping Procountor Oy create a modern, web-based accounting software that prioritizes user experience. Read the full story.



HPD Lendscape

HPD Lendscape successfully migrated its intelligent finance platform from Java Swing to Vaadin, a modern web framework. This migration was driven by Vaadin's pattern-driven approach and adjusted methods, enabling code reuse and scalability. By transitioning to Vaadin, HPD was able to transform its finance platform, making it web-based and enhancing its design and user interface. The article highlights the advantages of Vaadin's approach in enabling a seamless transition from Swing to Web, emphasizing code efficiency and improved user experience. Read the full story.



Summary

Modernizing Java Swing applications is essential for businesses aiming to meet current standards, stay competitive, and respond to users' growing expectations for a better user experience.

We encourage decision-makers to view modernization as a chance to improve their application's impact and operational efficiency. The migration process is greatly simplified with Vaadin's full-stack UI framework, Vaadin Flow, and automated tooling. The Modernization Toolkit makes this change easy and can be customized to fit your app's unique needs. The benefits? Better developer productivity, cost savings, modern features, good support, and stronger security – and above all – delighted users!

Get Started

Ready to bring your Swing applications to the future?

Let our modernization team guide you through a successful migration. Book your FREE consultation with our solution architects today and map your path to a successful migration.

Talk to an expert →

vaadin