Progress® OpenEdge®

Modern ABL Development Across the Software Lifecycle

WHITEPAPER

Modern ABL Development Across the Software Lifecycle

Modernizing applications developed with the Progress® OpenEdge® application development platform requires coordinated improvements across the software development lifecycle, from how developers write and test code to how teams evaluate quality, deploy releases and monitor application behavior.

When these practices work together, you can improve productivity, scalability and reliability while continuing to build on the Advanced Business Language (ABL) business logic that supports mission-critical operations.

To that end, modern ABL development focuses on strengthening the complete delivery process. AI-assisted development can help teams work more efficiently, but its output must be supported by sound application architecture, efficient database access, automated testing, measurable quality standards, repeatable delivery processes and runtime observability. Each capability addresses a different source of risk, and the greatest value comes from applying them as part of a connected modernization strategy.

This whitepaper examines seven areas that can help organizations advance that strategy:

1. ABL AI-assisted development and modern IDEs
2. Horizontal scaling
3. Query structure and optimization
4. Unit testing and shift-left development
5. Code quality and security
6. DevOps automation
7. Memory profiling

Used together, these practices will help you modernize your development process without abandoning the applications, data and expertise that you have developed over many years.

1 ABL AI-Assisted Development and Modern IDEs

AI is changing how software teams generate code, understand existing applications and complete repetitive development tasks. However, for developers that use the OpenEdge platform, a general-purpose coding assistant is insufficient. Not much ABL code is available in public repositories, generic large language models may lack the context needed to produce dependable results. They can generate code that appears plausible but fails to compile, targets the wrong OpenEdge application release or conflicts with established application patterns.

The OpenEdge AI Assistant addresses this gap by combining generative AI with knowledge that is specific to the OpenEdge platform. It provides ABL-aware assistance within Visual Studio (VS) Code and other AI-enabled development environments, including Cursor and Devin Desktop. The assistant can account for the OpenEdge application version being targeted, helping developers avoid unsupported language features and generate code that aligns more closely with the capabilities of their application environment.

The assistant can support a wide range of development activities. Developers can use it to generate or explain ABL code, analyze unfamiliar application components, create boilerplate, assist with modernization work and accelerate routine implementation tasks. It can also help teams work with existing applications by clarifying legacy code, identifying potential refactoring opportunities and suggesting approaches that they can evaluate against business and architectural requirements.

The OpenEdge AI Assistant includes OpenEdge Knowledge Base articles, broadening the context available when teams troubleshoot runtime and operational issues. Developers can use this knowledge when diagnosing application behavior, while database administrators and DevOps engineers can use the assistant to investigate configuration, deployment and database-related questions. This makes the Assistant relevant across a larger portion of the application lifecycle.

Organizations can further tailor responses through reusable rules and skills. Rules capture coding conventions, architecture requirements and preferred implementation patterns. The Skills feature group instructions and contextual guidance around particular tasks. By supplying the AI Assistant with organizational context, teams can improve the consistency of generated output and reduce the amount of information that must be repeated in every prompt.

AI-generated output still requires professional review. For example, developers are still responsible for validating syntax, business logic, security, performance and architectural fit. AI Assistant doesn't remove human judgment; it reduces the time required for repetitive work and information retrieval. Additional setup and usage guidance are available in the AI Assistant documentation for the OpenEdge platform.

2 Horizontal Scaling

Application growth can increase user activity, transaction volume and integration traffic. Addressing that growth by adding CPU or memory to a single server can eventually increase cost and constrain availability and capacity . Horizontal scaling offers another approach by distributing workloads across multiple application server instances.

Progress® Application Server (PAS) for OpenEdge® provides a foundation for horizontally scaled OpenEdge applications. For example, you can deploy multiple PAS for OpenEdge instances across physical servers, virtual machines, cloud infrastructure or containerized environments. As demand changes, you can introduce additional instances to increase capacity, without depending on a single application server.

Load balancing is central to this architecture. A load balancer directs requests across an available PAS for OpenEdge instance, helping distribute activity and reducing the likelihood that one instance becomes a bottleneck. A load balancer can also route traffic away from an unavailable instance in case of maintenance or unexpected failure.

The application must be designed to take advantage of this model. Stateless requests are easier to distribute, because any available instance can process them. Stateful designs may require session affinity or additional coordination, which can limit flexibility and complicate failover. It's important to evaluate how the application stores session data and accesses resources, and whether application components depend on a particular server instance.

Consistent configuration is equally important. The application should provision for each instance with compatible application artifacts, settings and dependencies, so every request behaves predictably, regardless of where it is processed. Automated deployment and configuration practices will also help reduce differences between instances and make it easier to add capacity reliably.

The application should validate horizontal scaling through realistic performance testing. Adding instances won't resolve inefficient queries, excessive memory usage or external system constraints. Testing helps determine whether workload distribution will provide the expected benefits and identify bottlenecks that you'll need to address elsewhere in the application stack. When you consider scalable architecture, load balancing and efficient application design together, you can support growth while improving resilience and deployment flexibility.

3 Query Structure and Optimization

Database performance is closely connected to the way applications retrieve and process data. Infrastructure improvements can provide additional capacity, but inefficient database access can continue to consume CPU, memory and network resources as workloads grow. That is why designing queries should be treated as an application development responsibility, not just a database administration task.

Indices allow the database to locate records efficiently without examining every row in a table. Queries that align with appropriate indices can reduce the work required to find matching records. Queries that can't use a suitable index may require broader scans, which increases response time and consumes more resources . The impact may be difficult to notice in development with a small dataset, but it will be substantial when the same query runs repeatedly against production data.

Another requirement is selectivity. The application should apply conditions that narrow result sets as early and accurately as possible, while using fields and operators that enable the database to use the intended indexes. Broad conditions, unnecessary sorting and redundant iteration waste processing resources, even when only a small fraction of the returned data is needed.

Applications should also retrieve only the information they need. Returning complete records or large datasets for a screen, service or report that contains just a few fields increases memory usage and network traffic. This becomes especially important when PAS for OpenEdge and the database communicate across different systems or environments.

AI-assisted analysis can help you identify common query patterns that deserve review, especially across large or unfamiliar code bases. You should evaluate these suggestions should using application knowledge, database statistics and representative performance testing. AI can accelerate investigation, but it does not replace human understanding of the data model, index strategy and expected workload.

Robust performance depends on cooperation between developers and database administrators. Developers determine how applications request data, while database administrators manage indexes, storage and operating conditions. By reviewing these factors together, teams can improve access without adding hardware to compensate for poor application performance.

4 Unit Testing and Shift-Left Development

Defects become more difficult and expensive to resolve as they move through the delivery process. Shift-left development addresses this problem by moving validation closer to the point where code is created. By incorporating testing into the development process, you can detect issues earlier, when the associated changes are still fresh and easier to correct, instead of relying on a separate quality assurance phase.

Smaller components, clear interfaces and limited dependencies are easier to test in isolation than tightly coupled procedures with an extensive external state. If you design for testability, you can improve application structure and make it easier to detect defects early.

ABLUnit tests provide a framework for creating and running automated tests for ABL applications, making it possible to organize tests into repeatable suites that verify expected behavior and help identify regressions when code changes. Information about code coverage can reveal which areas have been exercised and where additional tests may be needed, although you should consider coverage alongside the quality and relevance of the tests themselves.

AI can help reduce some of the effort involved in creating tests. For example, you can use AI to propose test cases, generate initial test structures or identify boundary conditions that need attention. These outputs still require review, because a generated test may reflect an incorrect assumption or confirm the implementation rather than the intended business requirement. However, if you use it carefully, AI can expand your testing capabilities without weakening accountability.

Automated tests are more valuable when they run consistently. Integrating ABLUnit testing into a continuous integration process enable tests to execute when you commit or prepare code for release. If the test fails, it stops the change from progressing until developers review the issue and prevents a known defect from moving downstream.

Shift-left development does not eliminate the need for testing later in the development process Integration, performance, security and user acceptance testing are still important for evaluating different risks. Still, unit testing helps to validate components early and repeatedly, so you can make changes with greater confidence.

5 Code Quality, Security and Development Standards

Functional correctness is only one way to measure application health. Code can produce the expected result but be complex and difficult to maintain, or vulnerable to security problems. As organizations modernize long-lived OpenEdge applications and increase the volume of code produced with help from AI, they need a consistent way to evaluate these broader characteristics.

Progress Application Quality & Security Management, or QSM, provides fact-based analysis of application quality and security. The service uses the Quality Scoring Method powered by Sigrid, the software assurance platform from the Software Improvement Group (SIG).

QSM provides measurable insight into maintainability, architecture and security. It helps organizations establish a baseline for an application, prioritize areas of technical debt and track whether quality improves over time. QSM also supports modernization planning by identifying complex or high-risk components that may require additional attention before integrating them into a new architecture.

Automated security checks help identify patterns in the code that need review prior to production, but they should be combined with secure development practices and human judgment. Teams are still responsible for determining whether a finding applies to the application context and for remediating any issues.

The OpenEdge AI Assistant can connect with quality analysis through the QSM Model Context Protocol (MCP) Server. This workflow can be used for newly generated code as well as existing code that is being reviewed or refactored. Developers can request analysis within their development environment and receive feedback on security and development standards while they work on the change.

Moving quality feedback closer to development enables earlier issue resolution and more consistent decision-making without shifting responsibility to the tool. Developers and architects must still validate business behavior and decide how to apply their findings. Combining measurable evidence with professional expertise depends on quality and security as continuous considerations during development.

6 DevOps, CI/CD and Automation

Modern development practices depend on the ability to build, test and deploy applications consistently, however, manual processes slow down development because they rely on individual knowledge and repetition, which leaves room for errors. DevOps practices reduce this risk by connecting development and operations through automated, shared processes and rapid feedback.

Continuous Integration and Continuous Delivery (CI/CD) organizes delivery activities into a repeatable pipeline. When developers commit code, automated processes compile the application, run ABLUnit tests, perform quality and security checks and prepare deployment artifacts. Each stage provides evidence that the change meets defined requirements before development continues.

By managing configuration through version-controlled files instead of undocumented manual changes, Configuration as Code (CaC) improves consistency and traceability. Infrastructure as Code (IaC) extends these practices to servers, networking and cloud resources. Together, they create environments that are easier to reproduce and restore while enabling horizontally scaled deployments by provisioning new PAS for OpenEdge instances with consistent settings.

Using containers to package an application and its dependencies can improve deployment consistency further. PAS for OpenEdge can be used in containerized deployments to create portable application environments and enable integration with orchestration platforms. Containers do not automatically modernize an application, but they can provide a repeatable operational unit that fits well with automated delivery and scaling practices.

You do not need to automate the entire lifecycle at once; a practical path may begin with automated builds and unit tests followed by quality checks, artifact management, deployment automation and infrastructure provisioning. Each step should eliminate a specific source of delay or inconsistency rather than introducing automation for its own sake.

The result is a delivery process that is more predictable and easier to improve. Earlier feedback, validated release workflows and shared operational knowledge increase consistency while reducing dependence on individual experience. Developers and operations teams can then focus on enhancing the application instead of repeating manual release activities.

7 Memory Profiling and Memory Management

Applications can be functionally correct and still become unstable if they do not use memory efficiently. As an application runs, it creates objects and allocates memory to support business processes and user activity. When memory is retained longer than necessary or allocation grows unexpectedly, response times slow and processes may eventually require manual intervention.

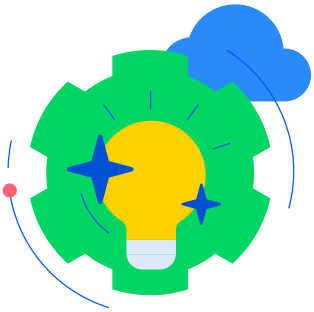
The OpenEdge Memory Profiler (OEMP) gives ABL developers visibility into memory allocation and object lifecycles within OpenEdge applications and PAS for OpenEdge environments. Developers can capture snapshots at different points in an application workflow and compare them to understand how memory usage changes over time.

Comparing snapshots can help isolate objects that continue to accumulate after a process should have released them. It can also reveal components that allocate more memory than anticipated. These findings narrow the investigation so that you can focus on the code paths, references and application events most strongly associated with increased memory usage.

Object lifecycle analysis helps identify references that prevent memory from being reclaimed. Even after the application completes its work, unintended references can cause memory usage to grow over time. Tracing where objects are created, retained and released helps to eliminate the underlying cause instead of reacting to its operational effects.

Memory profiling is useful during development, testing and troubleshooting. For example, you can profile a new feature before release, compare behavior before and after a refactoring effort, or reproduce a production scenario under controlled conditions. Repeated measurements provide stronger evidence than a single observation and can help determine whether a change actually reduces memory usage.

Finally, consider memory management as part of application observability and quality. Automated tests can confirm behavior while quality analysis can assess maintainability and security, and delivery automation can promote validated changes. Memory profiling adds insight into runtime resource usage, helping you to build OpenEdge applications that remain efficient and stable as they evolve.



Key Takeaways

- Modern ABL development is a lifecycle-wide discipline. AI-assisted development can accelerate coding, research and troubleshooting, but results depend on developer validation and the quality of the surrounding engineering practices.
- Scalable architecture, efficient queries and responsible memory management help applications perform reliably as workloads and deployment models change. Automated testing and fact-based quality analysis move feedback closer to development, so you can identify defects, technical debt and security concerns before they become more expensive to resolve.
- DevOps and CI/CD practices connect these controls to a repeatable delivery process, improving consistency across builds and releases.
- You do not need to replace proven OpenEdge applications to benefit from these capabilities. Instead, you can modernize incrementally by improving the way applications are developed, evaluated, deployed and observed. This helps build a stronger foundation for ongoing change that preserves trusted business logic while helping teams respond more quickly to new technical and business requirements.

Next Steps

Begin by exploring the OpenEdge AI Assistant to evaluate where AI-assisted development and OpenEdge Knowledge Base access can reduce repetitive work and support troubleshooting in your organizations.

If you are looking to strengthen shift-left practices, review the ABLUnit documentation and identify application components that would benefit from repeatable automated tests. Consider exploring Application Quality & Security Management to establish a fact-based view of maintainability, architecture and security, and evaluate the OEMP to gain greater visibility into memory allocation and object lifecycles.

For additional demonstrations and practical guidance across these topics, watch the Modern ABL Development for Today's OpenEdge Applications webinar on demand.

By selecting a focused starting point and connecting it to broader lifecycle improvements, your organization can modernize OpenEdge application development at a manageable pace while continuing to maximize the value of their existing applications and expertise.

Learn more about the Progress OpenEdge platform

About Progress Software

Progress Software (Nasdaq: PRGS) empowers organizations to achieve transformational success in the face of disruptive change. Our software enables our customers to develop, deploy and manage responsible AI-powered applications and personalized digital experiences with agility and ease. Businesses of all sizes get a trusted provider in Progress, with the products, expertise and vision they need to turn AI disruption into a competitive advantage. Millions of developers and technologists at hundreds of thousands of organizations depend on Progress every day. Learn more at www.progress.com

Worldwide Headquarters

Progress Software Corporation
15 Wayside Rd, Suite 400, Burlington, MA 01803, USA
Tel: +1-800-477-6473

facebook.com/progresssw
x.com/progresssw
youtube.com/progresssw
linkedin.com/company/progress-software
progress_sw_

Progress