

Playbook: From VB6 to .NET

Decision guide and migration path

Plenty of companies still run desktop applications built in Visual Basic 6. They work, but every year they get harder to maintain, harder to connect to other systems, and harder to staff. This playbook helps you decide what to do with your VB6 application and plan a phased move to .NET, without betting your operations on a single cutover.

1. Why it matters now

Microsoft handles the VB6 development environment and the VB6 runtime separately. According to its official support statement:

- The Visual Basic 6.0 IDE has been unsupported since April 8, 2008. There is no supported way to create or maintain VB6 applications.
- The runtime remains supported for the lifetime of supported Windows versions (the list includes Windows 11 and Windows Server 2025), but only for serious regressions and critical security issues in existing applications.
- The runtime is 32-bit only: on 64-bit Windows, VB6 applications are supported only in the WOW emulation environment.
- Microsoft does not support third-party OCX/ActiveX controls; that is up to each vendor.
- Microsoft itself strongly recommends replacing these applications with modern technology.

In practice, the application may keep opening, but every change is made with unsupported tooling, and the risk grows with each third-party dependency and each person who leaves the company.

2. Signs it is time to act

- Only one or two people understand the code, and nobody else is willing to touch it.
- You depend on OCX controls or DLLs from vendors that are gone or no longer respond.
- Every Windows, antivirus or hardware update raises fears that something will stop working.
- The business asks for integrations (web, mobile, APIs, e-invoicing, analytics) and the answer is "we can't" or a fragile workaround.
- Security or audit teams ask for centralized authentication, traceability or encryption the application cannot provide.
- Reports and reconciliations end up in Excel because the system cannot produce them.
- Installing the application on a new machine takes a procedure almost nobody remembers.

3. Options and when each one fits

There is no single right answer. The best option depends on how much the system is worth to the business, how much it will keep changing, and how well it is understood today. Options can be combined.

Retain and contain

When it fits: the system is stable, rarely changes, and its replacement is already scheduled. Isolate it (network and permissions), document dependencies, and test your backups.

Trade-offs: it does not reduce technical debt; it only buys time while you decide.

Rehost (move as-is)

When it fits: the urgent problem is hardware or the data center, not the code. The application and its database move to virtualized servers or remote desktops.

Trade-offs: it is still 32-bit VB6 with no supported IDE; a stepping stone, not a destination.

Wrap with APIs

When it fits: other systems need data or processes from the VB6 application, and the logic lives in the database or in reusable COM components. You expose .NET services on top of what exists.

Trade-offs: you get integration quickly, but add a layer to maintain, and VB6 stays.

Tool-assisted conversion (replatform)

When it fits: the code is reasonably clean and you want to reach .NET soon while keeping its structure. Tools translate forms and code to C# or VB.NET.

Trade-offs: the output usually carries over the same structure and debt; expect manual fixes and testing of every workflow, especially third-party controls, printing and reports.

Module-by-module refactor (strangler fig pattern)

When it fits: the system is critical, will keep evolving, and can run alongside a new one for a while. You build a .NET backend with APIs and a web frontend, one module at a time, until VB6 can be switched off.

Trade-offs: you need the source code and the ability to separate modules; two systems coexist during the transition and data must be coordinated.

Rewrite or replace (new build or SaaS)

When it fits: the business process has changed so much that the system no longer reflects it, or an off-the-shelf product covers it with reasonable adjustments.

Trade-offs: this is the option with the most uncertainty: undocumented rules are easily lost, and a one-shot delivery pushes value far out.

A combined path worth evaluating

Contain VB6 while you run the assessment, expose APIs for urgent integrations, and migrate module by module to an LTS release of .NET. Under Microsoft's official policy, LTS releases get three years of free support (for example, .NET 10 until November 2028), so also plan how you will keep upgrading.

4. A phased path

1 Assessment

Inventory of forms, modules, classes, OCX/DLL controls, reports, database, integrations and users. Interviews to locate critical business rules and the biggest pain points.

Deliverable: system map, risks, and a recommended option per module.

2 Pilot module

A small, visible module to validate the .NET architecture, data access, authentication, deployment, and how the teams work together.

Deliverable: a module in use by real users, plus lessons learned for the rest.

3 Incremental migration

Module by module following business priorities, with tests that compare results between the old and new systems, and clear rules on which system owns each piece of data.

Deliverable: each iteration adds migrated modules, documented rules and automated tests.

4 Decommission VB6

Once no user or process depends on the application: freeze changes, archive code and historical data according to your policies, uninstall, and remove dependencies.

Deliverable: decommission sign-off and documentation for the new system.

Illustrative example (fictitious)

Demo Distributors runs invoicing and inventory on a VB6 system backed by SQL Server. After the assessment, it starts with inventory lookup as a .NET and web pilot module on the same database. It then moves orders and invoicing, compares results between both systems at each release, and switches VB6 off once the last module is in use.

5. Risks and how to mitigate them

Risk	How to mitigate it
Business rules hidden in form events	Extract them and validate with key users before rewriting; characterization tests that pin down current behavior.
Third-party controls with no equivalent (grids, printing, reports)	Inventory them during the assessment, decide on replacements early, and test reports with real data.
Data migration	Profile data quality, use repeatable scripts, reconcile counts and totals, and rehearse the migration before each cutover.
Two systems on the same database	Define which system writes each piece of data, test transactions and locking, and monitor during coexistence.
Parallel run	Agree on a period in which both systems process the same work, compare results, and put the go-live criteria in writing.

Risk	How to mitigate it
Insufficient testing	Test cases based on real workflows, automation in the new system, and user acceptance testing.
Resistance to change	Involve users from the pilot onward, train them, and provide close support at each release.

6. Readiness checklist

- ☐ You have the complete source code and it builds in a controlled environment.
- ☐ There is an inventory of OCX/DLL controls, reports and integrations.
- ☐ You know which modules are critical and who their key users are.
- ☐ The important business rules are identified, even if not fully documented.
- ☐ Database backups are tested and a test environment exists.
- ☐ A business sponsor has time to set priorities.
- ☐ "Done" is defined for each module (acceptance criteria).
- ☐ There is a plan for running both systems side by side and for the final decommission.

7. How we approach modernization

Nextsoft does not migrate legacy systems on its own: we deliver modernization together with specialized strategic partners. The partner migrates the VB6 system; Nextsoft builds the new modules, integrations, frontend and QA, and is your single point of contact throughout the project.

Next step

Take the free legacy self-assessment: 10 questions on technology, operations, security and business, with the result calculated in your browser. If you would like to talk through your case, email us.

Self-assessment: nextsoftcorp.com/en/legacy-self-assessment

Email: contacto@nextsoftcorp.com

Sources (accessed September 2026)

Microsoft Learn, Support Statement for Visual Basic 6.0: <https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-basic-6/visual-basic-6-support-policy>

Microsoft, .NET support policy: <https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core>

Microsoft Azure Architecture Center, Strangler Fig pattern: <https://learn.microsoft.com/en-us/azure/architecture/patterns/strangler-fig>