Published by Betty Blocks | 2026

Betty Blocks

- Platform evaluations

IT's Guide to *AI Application Generation*

How to evaluate and select the right platform for building AI-powered business applications

Who this *guide* is for

This guide is written for CIOs, IT Managers, and Enterprise Architects evaluating platforms for building AI-powered business applications. It covers what to look for, how the major platform categories compare, and what questions to ask before committing to a vendor.

The scope is limited to platforms for building, deploying, and governing applications. It does not cover AI productivity tools used as assistants (Copilot, ChatGPT) or analytics and BI platforms.

CIO-focused content addresses governance and strategic tradeoffs. Enterprise Architect-focused content covers architecture, integration, and sovereignty in more detail. The guide can be read straight through or used section by section depending on where you are in the evaluation process.

Why most AI initiatives fail before they reach *production*

The gap between prototyping and production

95% of enterprise AI pilots fail to reach production. That figure, drawn from MIT research, does not reflect a shortage of ideas or budget. It reflects a structural gap between the tools available for prototyping and what is actually required to deploy AI-powered software in an enterprise environment.

The barrier to starting has never been lower. AI tooling has made it possible for non-technical users to build working prototypes in hours. The challenge is the next step: getting from a prototype to something that integrates with existing systems, passes security review, runs on real data, and can be maintained over time. Most of the platforms that make it easy to start were not designed for that journey.

The speed versus governance problem

IT leaders today face a genuine tension. The tools that generate AI applications quickly were built for developer speed and have no enterprise governance. The tools that have governance, primarily the established low-code platforms, were designed before the AI era and have added AI capabilities on top of an architecture that was not built for them.

The practical consequence is application sprawl. Business users build prototypes using consumer AI tools. Those prototypes move into operational use without security review, without an assigned owner, and without visibility from IT into what is running, what data it is accessing, or what it costs. This has become a meaningful governance risk for organisations that moved fast in 2024 and 2025.

Four underlying problems driving the platform decision

- **ROI from AI:** AI pilots that never reach production, initiatives disconnected from live business data, token costs that scale without measurable return.
- **Governance and Control:** application sprawl, AI-generated code without security review, no ownership or accountability for deployed applications.
- **Cost reduction:** SaaS licensing bloat, redundant applications across teams, vendor lock-in that limits flexibility and negotiating leverage.
- **Digital sovereignty:** data residency requirements, on-premises mandates, the need to own and port software without depending on a vendor's runtime.

The platform an organisation chooses will determine how well it addresses each of these.

The evaluation framework in Section 4 maps each dimension directly to these problems.

AI Application Generation: an emerging *category*

The evolution of application development

AI Application Generation is the most recent step in a progression that has been running for several decades. Each generation lowered the barrier to building software and introduced its own tradeoffs. Understanding which generation a platform belongs to is more useful than comparing features in isolation, because each generation has different architectural constraints, and those constraints determine what is possible at scale.

Generation	*What it changed*	*What it left unsolved*
Custom Development	Full control over architecture, code, and infrastructure. The baseline standard for enterprise software.	Slow and expensive to deliver. Expertise concentrated in developers. Knowledge leaves with people.
Low-code & No-Code	Visual builders made application development accessible to non-developers. Faster delivery, lower cost per application.	Proprietary models and vendor lock-in built in from the start. Not designed for the AI era.
AI-Assisted Low-Code	Existing low-code platforms added AI: code completion, logic generation, AI assistants within proprietary IDEs. Developers moved faster.	The platform architecture did not change. AI was layered on top of the same proprietary build process. Lock-in & runtime dependency remained.
AI Application Generation	AI generates the structural application model itself. Governance, ERP connectivity, and code portability designed in from the start.	Emerging category. Analyst frameworks still forming. Requires buyer education at the point of evaluation.

What AI Application Generation platforms actually generate

The distinction between AI Application Generation and other AI development tools is what the AI produces. Most AI development tools generate raw code. Every prompt edits source files directly. Successive changes frequently overwrite previous logic, break dependencies, or return incomplete output. There is no persistent structural model; the codebase is the only source of truth, and it degrades with each iteration.

AI Application Generation platforms generate a structural metadata model. That model defines the application: its pages, data structures, logic, and relationships. Code is compiled from the model on demand. When the application changes, the model changes; the code reflects it consistently. This means iterating on the application does not risk destroying what was already working.

Guided building versus true app generation

A related distinction is between guided building and true app generation. Some platforms use AI to assist a developer who is still manually navigating a visual builder: AI suggests components or generates fragments of logic within the platform's existing interface. That is guided building.

The AI is an accelerator; the platform architecture underneath is unchanged.
In true AI Application Generation, the AI produces the complete structural model from intent.
The developer is not configuring; they are describing.

The model is the source of truth, and the platform is built around generating and maintaining it. Guided building carries the architectural constraints of the platform underneath it. True AI Application Generation does not.

How to identify a true AI Application Generation platform

The distinguishing characteristic of AI Application Generation is that AI generates a complete structural metadata model from a single natural language prompt: pages, data structures, logic, and relationships, in one generation step. Not a skeleton, not logic fragments, not a template. A complete, production-ready application definition. That is a concrete, verifiable claim that can be tested in a demo.

How to evaluate AI Application Development *platforms*

The six dimensions below are used consistently across the competitive landscape (Section 5), the comparison matrix (Section 6), and the evaluation checklist (Section 8). Each maps to one of the four business problems from Section 2. Before beginning vendor conversations, rate each dimension as Critical, Important, or Nice-to-have. The dominant pain cluster will tell you where to concentrate the evaluation.

Dimension	*What to evaluate*	*Pain cluster*
AI Capability	Does the AI generate a structural metadata model or raw code? Is it true app generation or AI-assisted configuration? Does re-prompting rebuild the entire application or update the underlying model? Can non-technical users operate it independently from day one?	ROI from AI
Enterprise Governance	DTAP environments, versioning, sandboxes, rollbacks, CI/CD integration, role-based access controls, SSO, and audit trails. Can IT enforce deployment gates while business users build at speed?	Governance and control
Data Connectivity	Native connectors to your ERP systems (SAP, JDE, Workday, Salesforce). Real-time two-way sync. MCP Server support for AI assistants. No custom integration code required to connect to core systems.	ROI from AI
Total Cost of Ownership	Pricing model: per-app, per-user, per-token, or flat rate? External user charging Consumption risk. What does the cost look like at 3x and 10x current scale?	Cost reduction
Digital Sovereignty	On-premises, private cloud, or hybrid deployment options. Data residency compliance. Can generated code be exported and run independently of the vendor's platform and runtime?	Digital sovereignty
Time-to-Value	How quickly does a production-ready application reach deployment? What is the governance overhead before go-live? How does ongoing maintenance compare to custom development?	ROI from AI

Platform categories *compared*

Six platform categories are assessed below. Each is evaluated on its genuine merits before limitations are surfaced. All competitive claims are based on publicly available documentation and sourced research conducted in 2026.

Category	*Key players*	*Core strength*	*Core weakness*
Vibe Coding Tools	Lovable, Base44, Cursor, Replit	Speed to prototype, low barrier to entry	No structural persistence, no governance, full app rebuild on re-prompt.
AI Coding Agents	Claude Code, GitHub Copilot, ChatGPT	Developer productivity, broad capability	Developer-dependent; no deployment or governance layer.
Traditional Low-Code	Mendix, OutSystems	Enterprise governance, compliance track record	AI as IDE accessory; proprietary runtime lock-in; limited code export.
Platform Extensions	Power Apps, ServiceNow, Salesforce	Deep ecosystem integration	Vendor lock-in; external user licensing costs; limited cross-system scope.
AI Application Generation	Betty Blocks	AI-native architecture, governed generation, portable open-standard code.	Emerging category; buyer education required.
Custom Development	Internal teams, agencies	Full control, no vendor dependency	Slow to production; knowledge leaves with developers.

Vibe coding tools (Lovable, Base44, Cursor, Replit)

Vibe coding tools have changed what is possible for early-stage prototyping. A non-technical user can describe an idea and have a working interface within minutes. The category is improving rapidly and for the right use case it delivers genuine value.

What works with Vibe coding tools

- → Speed to prototype is unmatched. Idea to working interface in minutes, no technical knowledge required.
- → Low barrier to entry: no setup, no training, minimal cost to start.
- → Useful for validating concepts before investing in a production build.

Ideal for: concept validation, rapid prototyping, internal demos. Not designed for production applications requiring ERP connectivity, security review, or ongoing governance.

What does not work at enterprise scale with Vibe coding tools

- → No persistent structural model. Every prompt edits raw source files; successive changes risk overwriting logic, breaking dependencies, or generating placeholder code rather than functional implementation.
- → No DTAP environments, no RBAC, no CI/CD integration, no audit trail.
- → No native ERP connectivity.
- → Applications deploy to consumer hosting with no IT visibility into ownership, usage, or compliance.
- → Consumption-based pricing creates unpredictable cost at scale.

AI coding agents (Claude code, GitHub Copilot, ChatGPT

AI coding agents are developer productivity tools. For skilled developers they are genuinely transformative: they accelerate output, reduce boilerplate, and meaningfully change what a single developer can deliver in a day.

What works with AI coding agents

- → Significant productivity gains for experienced developers.
- → Broad language and framework support.
- → Works within existing development workflows and toolchains without disruption.

What does not work at enterprise scale with AI coding agents

- → Not an application platform. Generates code; does not deploy, govern, or maintain applications.
- → Requires a developer to own, manage, and maintain all output.
- → Non-technical users cannot operate independently.
- → No deployment layer, no DTAP, no ERP connectivity without custom integration work.

Ideal for:
developer-heavy teams looking to accelerate existing workflows. Not a substitute for an application platform where non-technical users need to build and own applications independently.

Betty Blocks MCP Server connects AI assistants like Claude or ChatGPT directly to live enterprise data with access controls intact and no custom integration required, extending the practical value of coding agents into enterprise data contexts.

Traditional low-code (Mendix, OutSystems)

Mendix and OutSystems have earned their position in enterprise IT over many years. Both offer mature governance frameworks and track records in regulated industries. For organisations with existing investments and trained teams, that history is a genuine asset.

What works with traditional low-code

- → Mature enterprise governance: DTAP environments, compliance tooling, audit trails.
- → Large certified partner ecosystems and implementation resources.
- → Strong track record in regulated sectors including financial services and public sector.
- → Both platforms are adding AI capabilities: OutSystems Agent Workbench, Mendix Maia & Maia Make.

What does not work at enterprise scale with traditional low-code

- → AI assists within a proprietary IDE; Maia Make produces microflows needing manual layout fixes.
- → Proprietary runtime lock-in: if a subscription ends, applications stop functioning.
- → Mendix offers no code export. OutSystems allows detachment only on subscription termination, outputting plain HTML, CSS, and JavaScript - no framework, and irreversible.
- → Per-app licensing penalises organisations building at scale.

Ideal for:
Best for organisations extending existing Mendix or OutSystems investments with incremental AI.

Not suited when AI-native generation, code portability, or cost predictability at scale matter.

Platform extensions (PowerApps, ServiceNow, Salesforce)

Power Apps, ServiceNow, and Salesforce Platform shine within their own ecosystems - offering pre-built connectors, familiar interfaces, and native identity alignment for applications that stay within a single vendor's world.

What works with platform extensions

- → Deep native integration within the vendor's ecosystem.
- → Pre-built connectors reduce time to first deployment.
- → Familiar to teams already using Microsoft, Salesforce, or ServiceNow tools.
- → Backed by large vendor organisations with enterprise support and SLAs.

What does not work at enterprise scale with platform extensions

- → Applications are locked to the vendor's runtime and cannot be moved or exported.
- → External user licensing is costly: Power Pages up to $200/month per 100 users; Power Apps Premium at $20/user/month regardless of activity.
- → Limited cross-ERP connectivity; designed to extend one platform, not bridge fragmented infrastructure.

Ideal for:
organisations building applications within a single ecosystem that do not require cross-ERP connectivity or portability.

Not suitable when multi-system integration, external users at scale, or infrastructure flexibility are requirements.

AI Application Generation platforms (Betty Blocks)

AI Application Generation is an emerging category defined by a metadata-driven architecture: AI generates a complete structural model from intent, governance is native to the platform, and the output is portable open-standard code. It is the category built to resolve the speed-versus-governance tradeoff that every other category requires IT to accept.

What works with AI Application Generation platforms

- → AI generates a complete structural metadata model, not raw code or logic fragments.
- → Governance, security, and deployment controls are native, not bolted on.
- → Native ERP connectivity without custom integration work.
- → Portable output: React and WebAssembly run independently of the vendor.
- → Flat-rate pricing with no per-app or per-user penalties at scale.

What to be aware of with AI Application Generation platforms

- → Emerging category: analyst frameworks still forming, buyer education required at the point of evaluation.
- → Narrower partner ecosystem than established low-code platforms today.

Ideal for:

IT leaders who need AI-speed application development with enterprise governance intact.

Organisations with ERP-heavy infrastructure, portability requirements, or significant application backlogs that cannot be addressed within existing platform constraints.

Custom development

Custom development gives complete control over architecture, code, and infrastructure. For applications with genuinely unusual requirements it remains the right answer.

What works with custom development

- → Full architectural control with no vendor constraints or runtime dependencies.
- → No platform licensing cost.
- → Fully extensible in any language or framework.
- → Complete code portability.

What does not work at enterprise scale with custom development

- → Production-ready applications take months to years to deliver.
- → Expertise is concentrated in specific individuals; knowledge leaves with them.
- → No built-in governance, security, or deployment infrastructure; everything must be assembled from scratch.
- → Maintenance burden grows with every application added to the portfolio.

Ideal for:
complex, mission-critical applications with requirements no platform can meet.

Not cost-effective for standard operational applications where delivery speed, governance, and predictable maintenance matter.

When to use which *platform*

Category	*Best suited for*	*Not suited when*
Vibe Coding Tools	Prototyping and concept validation	Production, governance, or ERP connectivity required
AI Coding Agents	Accelerating developer output	Non-technical users need to build independently
Traditional Low-Code	Existing platform investments; regulated governance needs	AI-native speed, code portability, or pricing at scale needed
Platform Extensions	Applications within one vendor ecosystem	Cross-ERP, external users at scale, or portability required
AI Application Generation	Enterprise AI applications combining speed with governance	Building core systems
Custom Development	Complex, bespoke requirements	Standard applications where speed and maintenance cost matter

Feature comparison across all *categories*

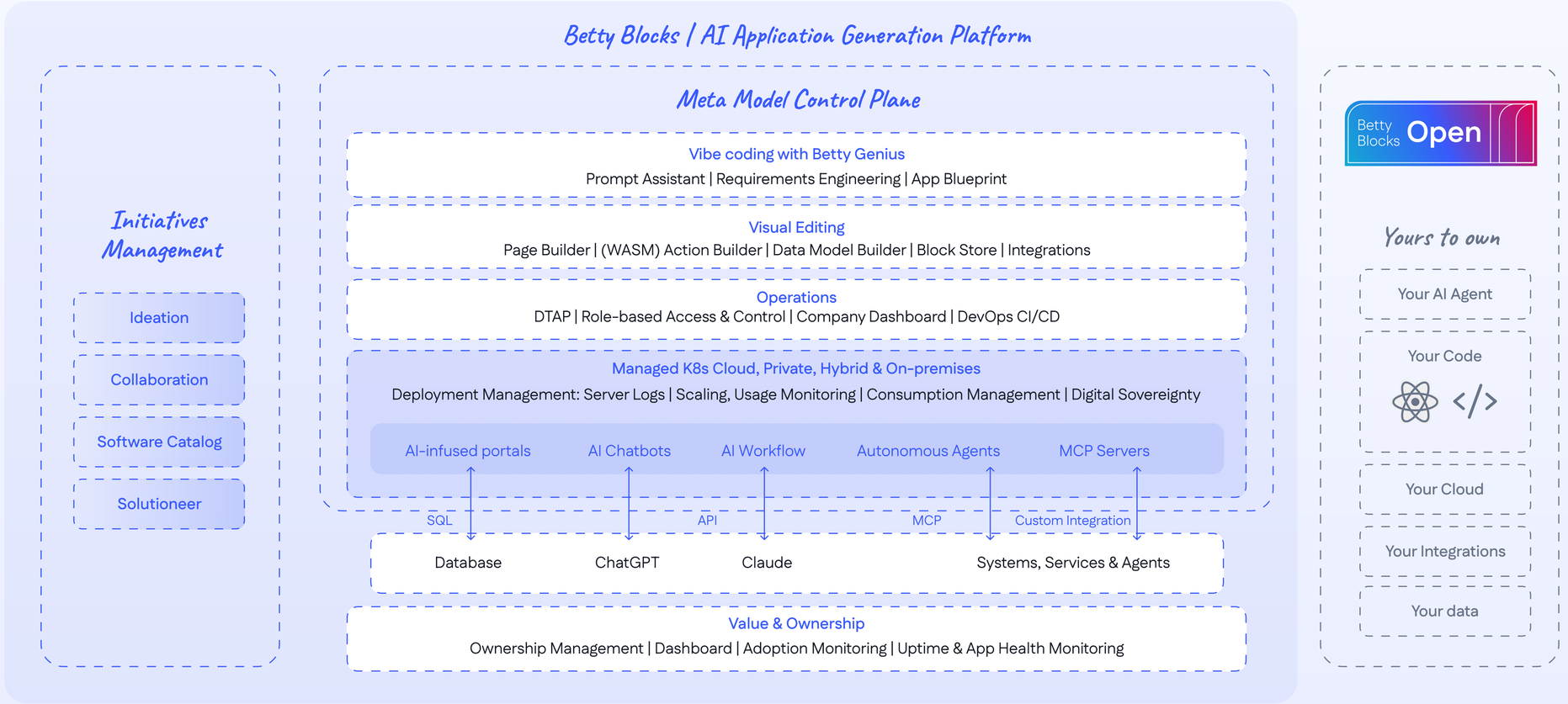
The table below describes each platform category's actual capability across eight dimensions. Each cell states what the platform provides and under what conditions. All assessments are based on publicly available documentation and sourced research from 2026.

	Vibe coding	*AI Coding Agents*	*Mendix/OutSystems*	*PowerApps/ServiceNow*	*Betty Blocks*	*Custom Dev.*
AI Generation Model	Raw code from prompts. No structural model. Re-prompting risks overwriting.	AI-generated code. No structural model.	AI assists in proprietary IDE. No complete structural model.	AI assisted. Localised logic only; no full application model.	Generates a complete metadata model from intent. Code compiles from on demand.	Developers integrate AI tools manually.
Enterprise Governance	None. No DTAP, RBAC, CI/CD, or audit trail.	Must be built and managed by the development team.	Native and established. DTAP, RBAC, CI/CD, compliance tools.	Available via add-ons. Not fully native.	Native. DTAP, RBAC, CI/CD, risk analysis, and audit logs included.	Achievable but manual.
ERP Connectivity	None. No native connectors.	Custom build required. Developers write and maintain all integrations.	Configurable. Connectors available; depth varies by system.	Strong for Microsoft stack. Limited beyond it.	Native connectors to SAP, JDE, Workday, Salesforce, Dynamics, and more. Two-way real-time sync.	Any integration possible. Significant development effort required.
Code Ownership & Portability	Source files owned by user. Tied to consumer hosting.	Full ownership. Deploys anywhere.	Locked to vendor runtime. No meaningful code export.	Locked to vendor ecosystem. No export.	Exportable React and WebAssembly. Runs independently.	Full ownership. Fully portable.
Ecosystem & Partner Maturity	Limited implementation partners.	Large developer community. No enterprise implementation partners.	Established. Large partner network.	Established. Extensive partner and ISV ecosystem.	Growing partner network.	Depends on the team or agency engaged.
Pricing Model	Consumption-based. Unpredictable at scale.	Seat or consumption-based. Variable at scale.	Per-app and per-seat. Inactive apps incur fees.	Complex pricing (per user/ per app/ extras). Gets expensive at scale.	Flat rate. No per-app or per-seat penalties.	Infrastructure, salaries, and maintenance costs. Variable at scale.
Time to First Production App	Prototype in minutes. Production requires developer and governance work.	Weeks for experienced developers. Dependent on skill and integration needs.	Months. Governance and compliance adds delivery time.	Weeks to months, depending on complexity.	Prototype in minutes. Production in weeks, with governance, ERP connectivity, and deployment controls built-in.	Months to years. Full build, test, and deployment.
Developer Extensibility	Limited. Code changes can be overwritten with prompts.	Fully open. Any language, framework, or toolchain.	Proprietary framework. Constrained to vendor standards.	Proprietary framework.	Open standards. Extends via React and WebAssembly.	Fully open. No constraints.

* AI Coding Agents score Partial for AI Capability and Time-to-Production when operated by skilled developers. For non-technical users, both dimensions score None.

The Betty Blocks AI Application Generation *platform*

Betty Blocks is built around three phases: Decide, Build, and Run. Most platforms start at Build - Betty Blocks starts at Decide. The diagram below shows the architecture, from initiative management through the Meta Model Control Plane to the Betty Blocks Open layer.



Initiative management: deciding what to build

Most platforms assume the right problem has already been identified. Betty Blocks addresses the step before that. A significant portion of the 95% AI pilot failure rate comes not from poor execution but from poor ideation: resources committed to the wrong problems, or to things already in the portfolio.

- **The Solutioneer:** an AI-powered ideation tool built into the platform. It guides stakeholders through structuring a business idea into a value document: problem statement, proposed solution, expected impact, cost estimate, and priority. This happens before any screen is designed or data model is generated, creating a structured, auditable business case for each initiative.

- **Software Catalog:** a centralised view of everything already in the application portfolio: what exists, who owns it, how much it costs, and whether it is in active use. This prevents redundant builds and surfaces reuse opportunities before development begins.

Together these create a prioritised backlog of AI initiatives that are worth developing, rather than a list of ideas that bypass any evaluation before consuming development resources.

The metadata model: how Betty Blocks generates applications

Every interaction with Genius AI updates a central metadata model that defines the application: its pages, data structures, logic, and relationships. Code is compiled from that model on demand. When the application changes, the model changes; the code reflects it consistently and without drift.
This is architecturally different from vibe coding, where prompts edit raw source files and successive changes risk overwriting what was already working.

It is also different from legacy low-code plus AI, where AI assists a developer navigating a proprietary visual builder. In Betty Blocks, the metadata model is the source of truth. The visual builder, the generated code, and the requirements documentation all reflect the same underlying model and are kept in sync automatically.

- **Genius AI:** a conversational AI assistant embedded throughout the platform. Generates complete multi-page applications from natural language, iterates on them conversationally, and updates the metadata model with every change. Unlike raw code generators, successive prompts refine the structural model rather than overwriting source files.

- **App Blueprint:** a real-time preview of the application's architecture before anything is built. Stakeholders review and adjust page structure, data models, and logic flows before committing to generation. This shifts the process from guessing to iterating.

- **Requirements Engineering:** converts natural language specifications into structured system requirements kept in perpetual sync with every subsequent iteration. When the application changes, the requirements documentation updates automatically, eliminating the documentation drift that typically makes enterprise codebases un-auditable over time.

Enterprise governance: IT controls what reaches production

Betty Blocks is the only AI Application Generation platform that integrates directly with existing enterprise CI/CD pipelines. Nothing reaches production without passing through the organisation's own approval process.

- **DTAP Street:** structured deployment environments (Development, Testing, Acceptance, Production) with parallel feature branching, controlled merging, and automated rollbacks. Business users build at speed; IT controls what is promoted and when.

- **Role-based Access and Controls:** custom roles, multi-layered permissions, and automated SSO provisioning. Permissions are defined at the organisation, application, and data level.

- **Standards-based DevOps:** direct integration with Kubernetes, ArgoCD, Azure DevOps, and GitHub. Betty Blocks deploys via standard container environments, fitting into existing SDLC architecture without custom pipeline work.

- **Risk Analysis and Developer Audit Logs:** automated scanning of application metadata for security vulnerabilities, technical debt, and outdated components; detailed developer activity logs for compliance review.

- **Company Dashboard and Software Catalog:** a single command centre for the full application portfolio: every app, its owner, usage, cost, and health in real time.

Data connectivity: native integration with core enterprise systems

Applications that cannot connect to live enterprise data cannot deliver operational value. Betty Blocks connects natively to the systems organisations run on, without custom integration code.

- **Native ERP connectors:** SAP S/4HANA, JD Edwards, Workday, Salesforce, Microsoft Dynamics, ServiceNow, Exact, and more. Two-way real-time sync: applications read from and write back to source systems.

- **Data Integrations:** consolidate disparate databases, web services, and CRM data into a single visual model. Eliminates data silos without ETL pipelines or bespoke sync scripts.

- **MCP Servers:** expose business data and system actions as tools for external AI assistants. Connect Claude or ChatGPT directly to live enterprise data with all existing access controls intact and no custom integration required.

Digital sovereignty: deployment flexibility and code ownership

Betty Blocks generates portable React frontends and WebAssembly backends. Both are open standards. Applications run on standard web servers without any dependency on the Betty Blocks runtime.

- **Code Export:** clean, human-readable React and WebAssembly code, downloadable at any point. Applications can be hosted and operated independently of Betty Blocks.

→ **WasmCloud and Kubernetes:** deploy on any infrastructure: public cloud, private cloud, on-premises, or hybrid. No proprietary runtime required.

→ **Data Residency:** host and process data wherever compliance requirements demand. No forced cloud regions or vendor-imposed data location constraints.

Pricing Model

Betty Blocks pricing is designed to scale without per-app or per-seat penalties.

→ **Unlimited applications:** flat-rate pricing. Building a larger portfolio of applications does not increase the base cost.

→ **No external user charging:** customer-facing and partner-facing portals deploy without per-user fees for people outside the organisation.

→ **Build and run tokens:** AI usage is metered on actual platform activity, not on user seats. Deployed applications that are not being actively modified do not generate ongoing AI cost

→ **CPI-based indexation:** price increases are tied to published inflation indices, not to vendor discretion.

Vendor evaluation *checklist*

The questions below are structured so that vendors who cannot meet enterprise requirements disqualify themselves on their own answers. Organise responses by vendor and score each against your dimension weighting from Section 4.

AI Capability

- [ ] Does the platform generate a structural metadata model, or raw source code?
- [ ] Is it true AI application generation (produces the full structural model) or AI-assisted building (AI helps a developer configure manually)?
- [ ] Does re-prompting or iterating trigger a full application rebuild, or does it update the underlying model?
- [ ] Can a non-technical user operate the platform independently from day one, without developer involvement?
- [ ] Can it generate a complete multi-page application including data models and logic from a single prompt?

Enterprise Governance

- [ ] Does it support DTAP environments with sandboxes, branching, merging, and rollbacks?
- [ ] Does it integrate with existing CI/CD pipelines (Azure DevOps, GitHub Actions, Kubernetes)?
- [ ] Is role-based access control included in the standard tier, or an add-on?
- [ ] Can IT see every deployed application, its owner, active users, cost, and health from a single dashboard?
- [ ] Does the platform include automated risk and security scanning, or does that require external tooling?

Data Connectivity

- [ ] Are there native connectors to core ERP systems without custom integration code?
- [ ] Does it support real-time two-way sync: read from and write back to source systems?
- [ ] Does it support MCP Server capability to connect AI assistants to live enterprise data?

Pricing and total cost of ownership

- [ ] Is pricing per-app, per-user, per-token, or flat rate?
- [ ] Are external users (customers, partners, vendors) charged separately?
- [ ] Is AI token consumption predictable and capped, or open-ended?
- [ ] What does the licensing cost look like at 3x and 10x current scale?
- [ ] Are there charges for applications that are live but not actively being modified?

Digital Sovereignty

- [ ] Can generated code be exported and run independently of the vendor's platform?
- [ ] Is the output based on open standards (React, WebAssembly) or a proprietary format?
- [ ] Does it support on-premises or private cloud deployment?
- [ ] What happens to deployed applications if the subscription ends?

Building the business *case*

Quantifying the cost of the current approach

Most organisations underestimate what their current approach costs, because the costs are distributed across teams and budgets rather than consolidated in a single view. A realistic current-state cost model should include:

- **Failed AI pilots:** internal time, external consultancy, and licensing for tools that did not reach production. Estimate the number of pilots run in the past 18 months and the average cost per pilot that did not deliver a result.

- **Application sprawl:** count applications in production with no identifiable owner, or owned by people who have left. Estimate hosting and support cost per application per year.

- **Redundant builds:** identify instances where two or more teams built similar tools independently. The cost is combined development time plus ongoing maintenance of duplicate systems.

- **Vendor lock-in risk:** for platforms where code cannot be exported, model the cost of migration if the vendor significantly raises prices or is acquired. This option value belongs in any long-term TCO calculation.

Modelling the return

- **Time-to-value:** AI Application Generation platforms deliver production-ready applications in weeks rather than months. Multiply the time saved per application by the number of applications in the backlog.
- **Portfolio scale:** flat-rate pricing means 20 applications cost the same as 5. Model the full backlog without a per-app penalty.
- **ERP connectivity:** applications connected to live ERP data in real time remove the lag and error rate of manual data extraction. Estimate the business value of operational decisions made on current rather than stale data.
- **Portfolio rationalisation:** centralised visibility over the application portfolio enables active cleanup. Retired licenses and consolidated applications are direct budget recovery.

The CFO conversation

The simplest frame for a CFO review: the total current-state cost of building and maintaining the application portfolio, versus the platform cost plus implementation. Current-state cost should include failed pilots, developer time, external consultancy, active licensing across all tools in use, and ongoing production maintenance. Platform cost includes the annual licence, implementation, and support. A Discovery Workshop structures this calculation as a facilitated engagement before any financial commitment. Betty Blocks runs these to quantify current-state cost together and model the business case with specific numbers.

Next *steps*

Where you are	*Recommended next step*
Early research	Work through the checklist in Section 8. Score each platform category against your dimension weighting from Section 4. Use the results to narrow the field before investing time in vendor conversations.
Activate evaluation	Request a Discovery Workshop. This is a structured facilitated engagement to map your current-state portfolio, quantify the cost of the status quo, and model the business case for a platform investment with specific numbers.
Ready to see the platform	Book a tailored demo. Demos are configured to your ERP stack, your use cases, and the evaluation dimensions that matter most to your organisation.

See how Betty Blocks fits your stack and business context

A focused walkthrough tailored to your ERP environment, use cases, and the platform questions that matter most to your team. No generic demo.

Book a walkthrough