
FIELD GUIDE · 2026

Business Central

in the Age of AI

A Field Guide for Developers and Consultants

47 chapters · 6 modules · complete AI code

Prompt engineering · Claude Code · GitHub Copilot

Azure OpenAI · RAG · AI agents · MCP servers

AUTHOR

Miloš Mikulášek

Business Central in the Age of AI

A Field Guide for Developers and Consultants

Miloš Mikulášek

2026 Edition · Version 1.0

Impressum

Business Central in the Age of AI *A Field Guide for Developers and Consultants*

© 2026 Miloš Mikulášek. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means — electronic, mechanical, photocopying, recording, or otherwise — without the prior written permission of the author.

Code samples included in this book are available at github.com/ai4bc-dev and may be freely used in your projects.

Author	Miloš Mikulášek
Web	ai4bc.dev
Email	milos.mikulasek@ai4bc.dev
GitHub	github.com/ai4bc-dev
Version	1.0
Published	2026
Language	English

Product and company names mentioned in this book may be trademarks of their respective owners. Microsoft, Business Central, Azure, GitHub Copilot, and Dynamics 365 are registered trademarks of Microsoft Corporation. Claude is a trademark of Anthropic, PBC.

Preface

This book was born out of a simple frustration: too many talented Business Central specialists spend too much time on things that AI could handle for them.

I've watched BC developers spend hours hunting for the right AL syntax for something Claude Code would write in thirty seconds. Consultants manually rewriting meeting notes into functional specifications, even

though AI could produce a first draft in two minutes. Freelancers turning down projects because they don't have time for documentation — even though AI would give that time back.

At the same time, I've seen the other group. People who embraced AI and wove it into their workflow. On specific routine tasks, they write code 3–5× faster. Specifications that used to take a day now take an hour. They offer their customers things the competition hasn't even seen yet.

This book is for the first group that wants to become the second.

Who This Book Is For

This book is written for three groups of readers who work with Business Central every day:

BC developers and programmers who know AL and want to systematically integrate AI into their development process. You'll learn to work with Claude Code and GitHub Copilot, call Azure OpenAI APIs directly from AL code, and build custom AI features for your customers.

BC consultants and functional analysts who write little or no code. You'll learn to use AI for creating functional specifications, test scenarios, user documentation, and customer communications — saving hours every week.

Freelance BC specialists who work solo or on small teams. You'll learn how AI acts as a “second developer” — helping with code, documentation, and communication, letting you compete with larger firms.

The book assumes a basic knowledge of Business Central and its ecosystem. You don't need to be an AL programming expert, but you should understand what BC does and how it works.

What This Book Doesn't Cover

This book is not an introduction to Business Central itself. It doesn't cover BC installation, accounting basics, or implementing standard BC processes. We assume you already know Business Central — and we teach you how to add artificial intelligence to it.

Nor is this book an academic treatise on machine learning theory. We don't care exactly how neural networks work inside GPT-4o. We care about how to use these tools in practice to save time and deliver more value to customers.

Prerequisites

To get the most out of this book, we recommend:

- At least a basic knowledge of Business Central (navigating the system, familiarity with the main modules)

- For chapters on AL programming: experience with AL or another programming language
- Access to a Microsoft 365 or Azure subscription (for hands-on examples in Module 3)
- VS Code with the AL Language Extension installed (for Modules 1 and 3)

Module 0 (Introduction and Orientation), Module 2 (AI for Consultants), and Module 5 (Action Plan) are accessible to anyone without programming prerequisites.

How This Book Is Organized

The book is divided into six modules (0–5) that build on each other logically — but each module can also be read on its own.

Module 0: Introduction and Orientation (3 chapters) explains why AI is changing the ERP world, what tools are available, and what realistic expectations look like. If you're just getting started with AI, start here.

Module 1: AI as Your Development Partner (11 chapters) covers everyday developer tools — prompt engineering, Claude Code, CLAUDE.md, GitHub Copilot, AI for testing, and a workshop where you'll build a complete AL extension from requirements to finished code.

Module 2: AI for the Consultant's Daily Work (9 chapters) covers using AI for the consulting and analytical day: specifications, test scenarios, documentation, customer communication, troubleshooting, SQL analysis, and data migration.

Module 3: AI Integration Directly in BC (10 chapters) is the technical core of the book. You'll learn to configure Microsoft Copilot in BC, call Azure OpenAI from AL code via HttpClient, and build three practical demo integrations — item categorization, AI-powered search, and intelligent suggestions. We also cover Power Platform AI Builder, Copilot Studio, GDPR considerations, and how to sell AI solutions to your customers.

Module 4: Advanced Techniques (6 chapters) covers RAG architecture with complete Python code, AI agents with an Azure Function implementation, and a decision framework for fine-tuning vs. prompting. It also explores MCP servers with a full TypeScript implementation, emerging tools, and a career roadmap.

Module 5: Action Plan (4 chapters) wraps up the book with concrete steps: a 30-day implementation plan with weekly checklists, how to convince your company to invest in AI with ROI calculations, how to keep learning without FOMO, and a final word of encouragement.

Appendices include a complete AL code reference from all chapters (Appendix A), a library of 12 prompt templates for common BC tasks (Appendix B), and 13 hands-on exercises for all modules (Appendix C).

Conventions Used in This Book

The book uses consistent typographic and textual conventions throughout.

Inline code denotes AL code, terminal commands, procedure names, variables, tables, and fields in the text.

```
// Code blocks look like this
procedure MyExample(): Text
begin
    exit('Working AL code, not pseudocode');
end;
```

All code samples are functional and testable in Business Central version 27 and later. The sample code namespace is **BCDemoAI**, with comments in English. This book targets BC v27. If you are running a newer release, all code remains compatible — check the What’s New section of the release notes for additional Copilot and AI features.

- **Note** marks supplementary information that’s good to know but isn’t essential for understanding the main text.
- **Warning** marks a security or GDPR consideration that’s important to respect in production deployments.
- **Tip** marks a practical shortcut or proven best practice from real-world projects.

Technical terms and abbreviations: The BC and AI fields use specialized terms (codeunit, prompt, deployment, embedding) that we introduce on first use and then use freely throughout the book.

Each chapter ends with a **Try It Yourself** section containing 2–3 hands-on exercises tied to the chapter’s content. Selected chapters are preceded or supplemented by a **Case Study** — an anonymized story from real BC practice.

Code Samples and Materials

All code from this book is available at github.com/ai4bc-dev. The repository contains:

- Complete demo AL extension (Purchase Approval with AI notifications)
- Azure OpenAI Helper codeunit (BC v27, fully functional)
- Python RAG indexer for BC data (Azure AI Search)
- TypeScript MCP server for BC OData API
- Python Azure Function for an email-to-order AI agent
- A set of prompt templates for developers and consultants (12+ templates)
- 12 SVG diagrams used in the book

Code samples from this book may be freely used in your projects without attribution. If you publish or share them, we appreciate a mention of the source, but it’s not required.

Feedback and Corrections

This book isn't perfect — no technical book in the AI space can be, because the field evolves faster than anyone can print. If you find a technical error, an outdated command, or any other issue, please let us know.

We also welcome feedback on what's missing from the book, what was most useful, or what specific BC+AI problems you're solving and would like to see covered in a future edition.

Contact: milos.mikulasek@ai4bc.dev

Acknowledgments

This book, in its scope and pace, would not have been possible without the very tools that are its main subject.

A big thank you goes to **Anthropic** for Claude — and specifically **Claude Code** and **Claude.ai**, which were present at every stage of creation: from the initial outline through writing and rewriting chapters to generating diagrams, code samples, and assembling the EPUB file itself.

Some of the text you're reading now went through dozens of iterations in dialogue with AI — and the result is better than it would have been without it.

There's a certain irony here: a book about how AI is transforming the work of developers and consultants was itself written with the help of AI. Perhaps that's the most honest proof that what we describe actually works.

Table of Contents

Module 0: Introduction and Orientation

1. Why AI Is Changing the ERP World (and Why It's Not Just Hype)
2. AI Tools for BC Developers — A Practical Overview
3. Realistic Expectations — What AI Can and Can't Do

Module 1: AI as Your Development Partner

1. AI Assistants Overview — Claude, ChatGPT, Copilot, Gemini
2. Prompt Engineering — Fundamentals for Developers
3. Prompt Engineering for the BC Context — Templates
4. Claude Code — Setup and Configuration

5. Claude Code — Working with AL Projects
6. CLAUDE.md — Context for BC Projects
7. GitHub Copilot — VS Code Setup for AL
8. GitHub Copilot — Effective Use with AL
9. Workshop — AL Extension with AI, Part 1: Requirements → Design
10. Workshop — AL Extension with AI, Part 2: Implementation → Review
11. AI for Testing AL Code

Module 2: AI for the Consultant's Daily Work

1. AI for Functional Specifications — From Notes to Document
2. AI for Test Scenarios
3. AI for Analyzing Customer Requirements
4. AI for Documentation — User and Technical
5. AI for Customer Communication
6. AI for Troubleshooting and Debugging BC Issues
7. AI + SQL for Analyzing BC Data
8. Module Summary — Your Personal AI Toolkit
9. AI for Data Migration into Business Central

Module 3: AI Integration in Business Central

1. Microsoft Copilot in BC: What It Does, How to Set Up, How to Extend
2. Azure OpenAI Service: Overview and Setup
3. Calling AI APIs from AL Code: HttpClient, JSON, and REST
4. Demo: Automatic Item Categorization with AI
5. Demo: AI-Powered Search in BC
6. Demo: Intelligent Suggestions for Sales Orders
7. Power Platform and AI Builder
8. Copilot Studio and Business Central Integration
9. Security, GDPR, and Ethics When Working with AI
10. How to Sell AI Solutions to Your Customer

Module 4: Advanced Techniques

1. RAG: Teaching AI Your BC Data
2. AI Agents: Automating Complex BC Processes
3. Fine-Tuning vs. Prompt Engineering: When to Use Which
4. MCP Servers and Custom Tooling for BC
5. Emerging Tools and Trends for BC Developers

Module 5: Action Plan

1. A 30-Day Plan for Bringing AI into Your Work
2. How to Convince Your Boss or Company to Invest in AI
3. How to Keep Learning Without Falling Behind
4. Conclusion: From Knowing to Doing

Appendices

A. Complete AL Code Reference B. Prompt Template Library for BC C. Chapter Exercises

About the Author

Miloš Mikulášek has worked with NAV and Business Central for over twenty years — as a developer, trainer, and someone who has been through every phase of the ecosystem, from the earliest versions of Navision to today's cloud-based BC. Beyond AL, he has SQL skills that let him solve problems where standard BC tools fall short.

He adopted artificial intelligence as a natural extension of his toolkit. He wasn't interested in the hype — he was interested in what AI can actually do in an environment he knows inside and out. The result is this book — *Business Central in the Age of AI* — and the community around it.

His approach is straightforward: no promises AI can't keep, no shortcuts where real system knowledge is needed. Just concrete tools and real-world examples from practice.

Find him at ai4bc.dev or at milos.mikulasek@ai4bc.dev.

Part I: Introduction and Orientation

Why AI is changing the ERP world and what tools are available

Why AI Is Changing the ERP World (And Why It's Not Just Hype)

Two years ago, I was standing at a BC conference, coffee in hand, chatting with a fellow developer. He was experienced — 12 years in BC, knew NAV going back to version 3.60. He said: “AI? It’s just hype. Nobody will be talking about it a year from now.”

Today that same person works at a company that told him to “implement AI in BC by end of year.” He’s standing in front of a whiteboard covered in sticky notes, with no idea where to start.

I don’t want that to happen to you.

This isn’t a chapter about how AI will change everything and how you need to immediately overhaul the way you work. This chapter gives you context. Realistic, practical context. What’s actually changing, why it affects BC developers and consultants specifically, and why now is the right time to start. Not because marketing says so — but because the data points to concrete things.

Let’s be honest about what AI content on the internet usually looks like.

On one side, the enthusiasts: “AI will replace developers!” “AI will change everything in five years!” “Prompt engineering is a new superpower!”

On the other side, the skeptics: “Just hype.” “Hallucinations are a problem.” “My customers don’t want AI.”

And then you go back to work, open VS Code, stare at the AL code — at a codeunit you need to finish by Friday — and think: “So what now? How does any of this actually help me?”

None of those articles told you how to write a codeunit with AI assistance. How to describe BC architecture to an AI. How to work with CLAUDE.md on a real BC project. How to integrate Azure OpenAI into AL code in a way that’s production-ready.

That’s the problem this book solves. And this chapter is the starting point — I’ll give you a framework to understand where we are, where we’re headed, and what it means for you specifically.

What’s Actually Changing — Data, Not Hype

Let’s start with numbers. GitHub ran a developer productivity study with GitHub Copilot — they tested 95 developers on specific tasks. The finding: an average 55% speedup on routine programming tasks. Fifty-percent. That’s not a negligible number.

McKinsey’s 2023 study reports that AI tools can increase developer productivity by 20 to 45 percent. We’re talking about tasks like boilerplate code, documentation, testing, and code review.

But here's the catch — these studies are based on general-purpose code. JavaScript, Python, Java. What about AL and Business Central?

Why BC Developers Have Both an Advantage and a Disadvantage

Let's get specific. Business Central occupies a unique position in the AI landscape.

The disadvantage: AL isn't represented in AI training data as well as mainstream programming languages. If you ask an AI to "write me an AL codeunit for X" without context, you'll get something that looks syntactically plausible but lacks knowledge of BC conventions, event architecture, and table relationships. AI doesn't know BC specifics like DataPerCompany, how extensions vs. the base app work, or how to properly subscribe to events.

The advantage: And here's the good news. Because you know what AL and BC mean, you have a huge edge over anyone using AI without that context. You know what the AI got right and what it got wrong. You understand the architecture. You're the domain expert — AI is your tool.

● **Note:** A good BC developer with AI will be more productive than an average BC developer without AI. And significantly more productive than an excellent general developer with AI but no BC knowledge. Your domain expertise is the key — AI amplifies it, it doesn't replace it.

Three Waves of AI in the ERP World

I like to think about where we are through a metaphor of three waves. It helps you understand our current position and what's coming next.

Wave 1 — AI assistants for developers. These are tools like ChatGPT, Claude, and GitHub Copilot. They help you write, explain, and debug code, generate documentation, and answer technical questions. This is where we are now. This wave is real, accessible, and usable today. Modules 1 and 2 of this book cover it in detail.

Wave 2 — AI built directly into the product. Microsoft Copilot in Business Central, AI-powered search, automatic suggestions for users, and intelligent workflows. We're beginning to enter this phase. Microsoft is actively developing it, and recent BC releases include the first features. Module 3 focuses on this.

Wave 3 — Autonomous AI agents. Systems that monitor processes on their own, detect anomalies, and propose or execute actions within BC. This wave is on a two- to three-year horizon. We discuss it in Module 4 because we want you to have the context — even though we can't fully implement it yet.

Why am I telling you this? Because understanding where we sit on the timeline helps you set the right expectations. Waves 1 and 2 are here and working. Wave 3 is coming — and those who prepare today will be ready tomorrow.

What This Means for Your Career

Let me tell you what I honestly think. The good news: BC and AL expertise remains essential. AI won't replace it anytime soon. Customers still need someone who understands their business, who designs solutions, and who takes responsibility for the outcome.

● **Warning:** A developer who doesn't use AI will spend all day on work that an AI-assisted developer finishes by lunch. That doesn't mean AI will replace you — but you could be replaced by a developer who uses AI better.

And that developer doesn't have to be as experienced as you — because AI helps bridge part of the experience gap.

So: the question isn't whether to use AI. It's how to use it well. And that's exactly what this book will teach you.

Summary

Let's recap what we've covered.

First: AI in BC development isn't hype. There are real numbers behind it — a 55% productivity gain in the GitHub Copilot study, a 20–45% productivity increase according to McKinsey.

Second: BC developers have a unique advantage. We understand the context that AI doesn't know on its own. We're domain experts — AI is a tool that amplifies that expertise.

Third: We're in Wave 1 — AI assistants are here today, accessible and usable. Wave 2 (AI in the BC product) is starting. Wave 3 (autonomous agents) is on the horizon.

Fourth, and most importantly: It's not a question of "whether." It's a question of "how well." And that's exactly what the rest of this book focuses on.

Real-World Examples: AI in BC Practice Today

To move beyond numbers and studies, let's look at concrete situations from BC specialists who already use AI.

Example 1: A Developer and Boilerplate Code

A developer gets this assignment: "We need a table for tracking service contracts with fields for customer, start date, end date, monthly fee, status, and notes. Plus a page for management and a codeunit for automatic expiration checks."

Without AI: Open VS Code, create the table, field by field. Then the page, field by field. Then the codeunit. Total: 45–60 minutes of routine work with no real decisions — just writing standard structure.

With AI: Submit a prompt with the requirements and conventions (prefix, comments), and in 5 minutes you have a complete table, page, and codeunit skeleton. Another 15 minutes on review and customization. Total: 20 minutes. Savings: 30–40 minutes on a single task.

Example 2: A Consultant and a Customer Workshop

A consultant has a discovery workshop tomorrow at a customer’s site. She has the topic (changes to the shipping process) but no agenda. Previously, she’d spend an hour preparing.

With AI, in 15 minutes she has: a structured workshop agenda, key questions ranked by priority, a list of documents the customer should bring, and an overview of common BC issues in this area. She arrives at the workshop prepared — and the customer notices.

Example 3: A Freelancer and a Proposal

A freelancer receives an inquiry about a BC extension for lease management. He needs to write a proposal with estimates, scope, and pricing.

With AI: he inputs the requirements and the AI breaks the project into components, estimates hours for each, identifies risks, and proposes implementation phases. The freelancer edits the proposal, adds his rate, and in 30 minutes has a professional document — instead of 2 hours of thinking and formatting.

What These Examples Have in Common

In all three cases, AI doesn’t make the decisions — the person does. AI handles the routine work: structuring text, generating boilerplate code, formatting documents. The person adds expert knowledge: BC architecture, customer specifics, pricing strategy.

This is the model that works. And it’s exactly the model the rest of this book will teach you.

Try It Yourself

Exercise 1: Measure your baseline. This week, write down 3 tasks you do and how many minutes each one takes. By the end of the week, you’ll have a benchmark to compare against when you start using AI.

Exercise 2: First contact with AI. If you haven’t tried AI yet, open claude.ai or chatgpt.com and enter: “I’m a BC developer. Write me a simple AL procedure that checks whether a customer has an email address filled in.” Does it work? What would you change?

AI Tools Overview for BC Developers

A colleague recently told me he'd spent an entire weekend testing AI tools. Claude, Cursor, Codeium, Perplexity, ChatGPT... He installed them, tried them out, and read the reviews.

On Monday he came to work and had no idea which one to actually use for his BC project. He went back to Google.

That's the exact opposite of what we want. The goal isn't to have as many tools as possible. The goal is to have the right tool for the right situation and learn to use it well.

The problem isn't a shortage of AI tools. There are too many — and new ones appear every week. ChatGPT, Claude, Gemini, Copilot, Cursor, Codeium, Tabnine, Perplexity... each with its own marketing campaign about why it's the best.

For BC developers, this creates analysis paralysis. How many tools do I need? Do I have to pay for all of them? What does the rest of the BC community use?

The answer is simpler than you think. There are three categories of tools relevant to BC work. In each category, there are one or two clear winners. And by the end of this chapter, you'll get a decision framework — one simple rule to determine what to use for any given situation.

Category 1: General-Purpose AI Assistants

This category includes AI tools you interact with through a chat interface. You type questions, get answers, submit code for review, and ask for documentation. They're multi-purpose tools.

Claude by Anthropic is currently my primary tool for BC work. Why? It has one of the largest context windows available — meaning you can feed it an entire large AL file or a full specification and it handles it consistently. For writing functional specs, analyzing requirements, documentation, and generating AL code with a good prompt, it's excellent. The free tier gives you the basic version; Claude Pro at \$20/month gets you a better model and higher limits. And Claude Code — we'll cover it in Module 1 — is a full development assistant built on Claude.

ChatGPT by OpenAI was the first mass-market AI assistant and is still very good. For quick questions, concept explanations, and general coding, it works great. For BC-specific work, Claude is slightly better thanks to its larger context window, but ChatGPT is a solid alternative — especially if you're already paying for it or have it through a corporate license.

Gemini by Google is interesting mainly for those who work in the Google Workspace ecosystem. For coding, it's currently a step behind Claude and ChatGPT, but it has useful features for document analysis and real-time search.

Verdict for BC work: I recommend Claude as your primary tool. ChatGPT as a backup, or if your company already provides it.

Category 2: Developer AI Assistants (In-Editor)

This is the category where AI integrates directly into VS Code or another editor and helps you while you write code in real time.

GitHub Copilot is the most widely used tool in this category. It works inside VS Code, sees your currently open files, and suggests code completions. For general programming, it's excellent. For AL, it depends heavily on context — if Copilot can see good AL files in your workspace, the suggestions improve significantly. Since November 2024 there's a **Free tier** (2,000 completions + 50 chat messages per month), enough for the first BC experiments. The paid Individual plan is \$10/month, and it's free for students and open-source projects. In Module 1, we'll look at setup and effective usage for AL in detail.

Claude Code is a different kind of tool — it's not just inline code completion. It's a full-stack AI development assistant. It reads your entire project, makes changes across files, runs commands, and understands git history. I call it “an AI developer sitting next to you who sees everything you see.” For AL projects, it's currently the most powerful option. Included with Claude Pro. Lessons 1.4 through 1.6 are entirely dedicated to Claude Code for BC projects.

Verdict: Claude Code as your primary tool for deeper project work, GitHub Copilot as a complement for inline suggestions while coding.

Category 3: The Microsoft AI Ecosystem for BC

Here we get into territory that's specific to BC developers and consultants.

Microsoft Copilot in Business Central is an AI assistant built directly into the BC user interface. It helps users search records, explains fields and functions, and suggests actions. From a developer's perspective, what's interesting is that this Copilot can be extended with custom AL add-ins — and that's a major opportunity for BC partners. Customers will want it. Covered in detail in Module 3.

Azure OpenAI Service is what you'll need when you want to integrate AI directly into a BC solution for a customer. It's a GPT model accessible via API, hosted on Azure. Advantages over calling the OpenAI API directly: it's enterprise-grade, GDPR-compliant, comes with an SLA, and your data isn't stored by the provider. For production BC solutions, this is the standard choice. In Module 3, I'll show you how to call it directly from AL code via HttpClient, with working examples.

AI Builder in Power Platform is the no-code path to AI capabilities. Document recognition, sentiment analysis, predictions, and invoice data extraction. No code required — through Power Apps or Power Automate. For BC consultants who don't want to go the AL route, it's a viable alternative. Also covered in Module 3.

Decision Framework — When to Use What

Let's boil it down to a simple rule. Here's a table to help with everyday work situations.

Situation	Recommended Tool
Writing AL code and want AI as a partner for the whole project	Claude Code
Want inline code completion while typing in VS Code	GitHub Copilot
Quick question, explanation, debugging a simple issue	Claude or ChatGPT
Writing a functional spec or documentation	Claude
Integrating AI into a customer-facing BC solution (production)	Azure OpenAI Service
Customer wants AI features without AL development	AI Builder / Copilot Studio
Want to extend Microsoft Copilot in BC	Copilot Extensions (AL)

This framework covers 80% of situations. There will be exceptions, but it's a solid starting point.

Summary

Three categories of AI tools for BC work.

Category 1 — General-purpose assistants: Claude as primary, ChatGPT as an alternative. Both via chat interface, excellent for documentation, specs, and code with a good prompt.

Category 2 — Developer assistants in the editor: Claude Code for in-depth project work, GitHub Copilot for inline suggestions. These two tools will be closest to you during everyday development.

Category 3 — The Microsoft ecosystem: Copilot in BC and Azure OpenAI for production integrations into BC solutions. Power Platform for no-code approaches.

Recommended starter stack: Claude Pro at \$20/month and GitHub Copilot at \$10/month (or start with the Free tier). For \$30/month — less than half a day's billable work — you have a full AI toolkit for BC development. For heavy Claude Code users, the **Claude Max plan** (\$100/month) offers significantly higher limits and is typically the better choice for professionals who rely on AI for most of their coding work.

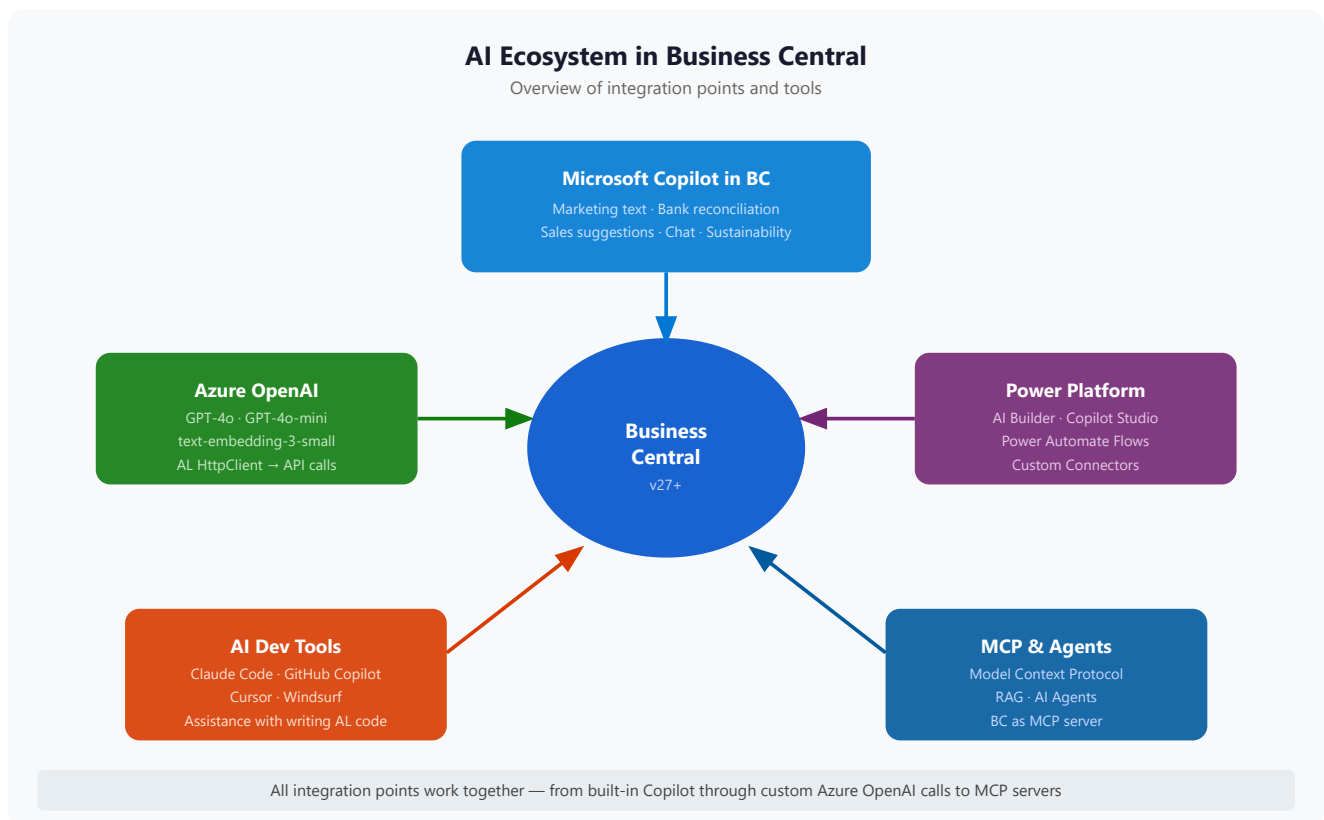


Fig. 1: AI Tools Ecosystem for BC

When to Use Which AI Tool in a BC Project

Decision matrix for BC developers and consultants

Tool	Best For	Specific BC Use Cases	Not Suited For
Claude Code / GitHub Copilot AI IDE assistant	Writing and refactoring code Debugging and explaining Generating tests Code review	AL codeunits and page extensions Fixing AL compile errors Documenting procedures CLAUDE.md for BC projects	Production AI features in BC for customers
Azure OpenAI + AL HttpClient Custom integration	Custom AI logic in BC Categorization and classification Generating text from BC data Structured JSON output	Auto-categorization of BC items Sales item suggestions AI-powered search Intelligent alerts	Writing code, generating developer documentation
Microsoft Copilot in BC Built-in Copilot	Ready-made AI features, no code Quick deployment at customer Standard Microsoft UX Minimal setup	Marketing text for items Bank reconciliation Order line suggestions Sustainability tracking (BC 27)	Custom business logic, company-specific processes
Power Platform AI Builder Low-code AI	Document processing AI automation without code Customer chatbots Predictive models	Invoice scanning into BC Customer bot for order status AI classification via Power Automate Demand forecasting	Complex custom code, high-volume API calls
RAG / Agents / MCP Server Advanced techniques	Complex knowledge systems Autonomous processes in BC Direct AI access to BC data	Chatbot over BC documentation Agent for email processing BC as MCP server for Claude	Simple tasks — unnecessarily complex solution

Fig. 2: AI Tools Decision Matrix

Realistic Expectations — What AI Can and Can't Do

One of my colleagues — an experienced BC developer with 8 years under his belt — spent three hours trying to get ChatGPT to write a complete BC module for contract management.

The result? Code that looked correct and compiled. But roughly half of it didn't work as intended. Relationships were wrong. Event subscribers were missing. Naming conventions didn't match their project. He spent another hour fixing things, then gave up.

He walked away frustrated and convinced that “AI isn't up to BC development.”

He was wrong. But his expectations were off. And that's exactly the problem this chapter addresses — before we dive into hands-on work in Module 1.

AI tools are extremely powerful — but only if you know what to use them for.

If you expect more than they can deliver, you'll end up frustrated like my colleague. You'll say “AI doesn't work” — even though the problem was in how you used it, not in the tool itself.

Conversely, if you expect too little, you'll leave a huge amount of saved time on the table. You'll use AI only for trivial tasks, when it could save you hours on more substantial work.

Properly calibrating your expectations is literally the difference between “AI does nothing for me” and “AI saves me 10 hours a week.” Let's get that calibration right.

What AI Does Well — for BC Developers and Consultants

Boilerplate and routine code. Codeunits for validations, simple calculations, standard CRUD operations, event subscribers for standard events, AL interfaces and their implementations — AI handles these very well, provided it gets a good prompt with context. In Module 1, you'll learn exactly how to write that prompt.

Explaining existing code. “Explain what this codeunit does” or “Why might this code be causing error X” — this is an area where AI excels. It's useful when onboarding to a new project, during code review, or simply when you've inherited code from a colleague who's no longer at the company.

Documenting code. Adding comments to existing code, generating XMLdoc documentation for procedures, writing a README for an extension — AI does this quickly and well. For consultants: an hour's worth of documentation work shrinks to 10–15 minutes.

Writing functional specs and test scenarios. This is where consultants save the most time. The customer tells you what they want — you jot it down in notes — AI turns it into a structured functional spec. Or the reverse: it generates test scenarios from a spec. We'll walk through this workflow in detail in Module 2.

Refactoring and code review. “How could I write this code better?” or “What are the potential issues in this code?” These are questions AI answers well. It can flag performance problems, missing error handling, or inconsistent naming.

Debugging with a problem description. “I’m getting this error in AL: [exact error]. Here’s the code: [code]. What’s wrong?” This works surprisingly well — especially for compile-time errors and logical issues that can be described in text.

What AI Doesn’t Do Well — or at All

A complete BC module from scratch, without detailed context. This is exactly what my colleague from the intro tried. “Write me a contract management module” is too vague. The AI doesn’t know how your company tracks contracts, what the business rules are, or how it integrates with other modules. Without that context, it generates a generic solution that looks right but doesn’t fit your project.

The fix: break it down. One codeunit at a time. With the context of tables, business rules, and your project’s naming conventions.

Understanding the customer’s business logic. AI doesn’t know that your customer has a 15-year-old invoicing process that’s documented nowhere but everyone knows. AI doesn’t know about the exceptions that “have always been that way.” AI doesn’t know about political decisions (“this process exists because the CEO wants it that way”). Only you know that — and that’s why you’re irreplaceable as a BC specialist.

Testing code in Business Central. AI writes code, but it doesn’t run it. It won’t check whether an extension compiles. It won’t catch a runtime error. Debugging in BC — that’s still on you. AI can help you design a solution, but verifying actual behavior in a BC environment is your job.

Replacing BC ecosystem knowledge. This is critical. A developer with no AL or BC knowledge will get equally unusable results from AI as they would without it. AI amplifies your knowledge; it doesn’t replace it. That’s why this book assumes you already know the basics of AL and BC.

Guaranteeing up-to-date information. AI models have a training cutoff — a date beyond which they have no data. The latest BC version, new API functions, current AL best practices — always verify against official Microsoft documentation. AI can tell you how things worked up to version 26, but it might not know what changed in version 27.

The Golden Rule of Working with AI in BC

AI is a junior developer who has never worked in Business Central.

It’s smart, hardworking, and fast. It reads everything you give it. It does what you tell it. But it has never seen a real BC project. It doesn’t know your customer’s processes. It lacks the intuition that comes from years of experience in BC.

That means it needs: - Project context (that’s what CLAUDE.md is for — more in Lesson 1.5) - Clear instructions with enough detail - Your review of the output - Correction when it makes a mistake — and it will

● **Warning:** Always review AI-generated code the same way you'd review code from a junior colleague. Never assume "AI surely got it right." Compiles doesn't mean correct.

Most Common Mistakes I See from Beginners

Mistake 1: Prompts that are too vague. "Write me a BC extension." The AI doesn't know what, why, for which tables, or with what logic. Result: generic code that doesn't match your project. Fix: In Module 1, Lessons 1.2 and 1.3, you'll learn prompt engineering specifically for the BC context.

Mistake 2: Not reviewing the output. AI can generate code that compiles but does the wrong thing. It overwrites records it shouldn't. It references tables that don't exist in your BC version. Review code as if it came from a junior.

● **Tip:** Overly long conversations degrade output quality — AI "forgets" context from the beginning. For a new task, always start a fresh conversation with fresh context.

Summary

AI in BC work excels at boilerplate code, explanations, documentation, specs and test scenarios, refactoring, and debugging with descriptions. These are the areas where you'll save hours every week.

Don't expect AI to deliver complete solutions without context, understand your customer's business logic, test in BC, or guarantee up-to-date information.

The golden rule: a junior developer with no BC experience. Give it context, review the output, correct mistakes.

Practical Test: Where AI Saves Time and Where It Doesn't

We ran an informal survey of 15 BC developers and consultants who actively use AI. The question: "Where does AI save you the most time, and where is it useless?"

Top 5 Time-Saving Areas (Ranked by Votes)

1. **Generating boilerplate AL code** (14/15 votes) — tables, pages, event subscribers, getters/setters. AI handles this perfectly because it's a structured pattern.
2. **Writing documentation and specs** (13/15) — structure, wording, consistency. AI drafts are about 80% usable.
3. **Explaining unfamiliar code** (12/15) — legacy code, third-party extensions, undocumented modules. AI reads and explains faster than Stack Overflow.
4. **SQL queries** (11/15) — BC's table structure is extensive. AI knows the correct table and field names and generates working queries.

5. **Debugging with error messages** (10/15) — exact error text + context = fast diagnosis. Better than Google.

Top 5 Areas Where AI Doesn't Help (Ranked)

1. **Customer business logic** (15/15 votes) — AI doesn't know the customer, their processes, or their exceptions. Only you do.
2. **Testing in a BC environment** (14/15) — AI writes code but doesn't run it in BC. Runtime verification is on you.
3. **Architectural decisions in a customer context** (12/15) — “Should we build this as an extension or a modification?” depends on the customer's situation.
4. **Licensing and commercial questions** (11/15) — AI doesn't know what pricing your partner offers.
5. **Debugging race conditions and intermittent errors** (10/15) — problems that can't be described in text require hands-on diagnostics.

Notice the pattern: AI excels at structured, repeatable tasks. It's weak at contextual decisions that require knowledge of the customer or the environment.

Try It Yourself

Exercise 1: Build your own list. Write down 5 tasks from your week. For each one, estimate: will AI save 50%+ of the time? 10–50%? Or not help at all?

Exercise 2: Test the golden rule. Give AI a task from your work as if you were assigning it to a junior colleague. What instructions do you need to add to make the result usable? Those are exactly the instructions that belong in your prompt.

Part II: AI as Your Development Partner

Prompt engineering, Claude Code, and GitHub Copilot

AI Assistants Overview — Claude, ChatGPT, Copilot, Gemini

Developers in the BC community keep asking the same questions over and over: “Should I use Claude or ChatGPT? GitHub Copilot or Cursor? Is Gemini better for code?” And the honest answer is — it depends on what exactly you’re doing. But that diplomatic answer doesn’t help when you need to start working today.

This chapter gives you a straight answer. No diplomatic “it depends on the situation” — concrete recommendations for BC developers, with real-world examples. You’ll see the results of the same test across four tools, and by the end you’ll know what to install and what to skip.

AI tool benchmarks on the internet test Python, JavaScript, general algorithms, and LeetCode problems. Nobody tests how tools perform with AL code, BC architecture, event subscribers, table extensions, and SMTP Mail Management.

And that’s exactly the problem. You don’t need to know who wins at Python. You need to know what works for your job. Not for a generic developer writing web APIs in Node.js. For you — a BC developer who works with AL, codeunits, tables, and business logic in the Microsoft ecosystem.

That’s why we prepared our own test on a real BC task.

The Same Test for Every Tool

I gave each tool an identical prompt. Here’s the exact wording:

“I’m a BC developer. I need to write an event subscriber that, after posting a sales order (OnAfterPostSalesDoc), checks whether the order value exceeds 10,000 CZK and if so, sends an email notification. BC version 27.”

This is a real-world task. Exactly the kind you’d get from a customer.

Claude (claude-sonnet-4-6)

Claude understood the context immediately. It identified the correct event — and correctly noted that for a post-posting workflow, OnAfterPostSalesDoc is typically used, but the exact moment you want the notification matters, and offered alternatives. It used correct AL syntax for v27, added proper table extensions for accessing the Sales Invoice Header, and hooked into SMTP Mail Management for email. The code was functional with minimal edits — I accepted it into the project after one minor naming convention fix.

The key advantage that sets Claude apart from the others: When you give it a CLAUDE.md file describing the project — we’ll cover this in detail in Lesson 1.5 — the output quality improved dramatically. It automatically started using our naming conventions, recognized which tables we had in the project, and adapted the code to the specific situation. Instead of a generic solution, you get one that genuinely belongs in the project. The other tools don’t do this as naturally without explicit context.

Vážení čtenáři, právě jste dočetli ukázkou z knihy ***Business Central in the Age of AI***.
Pokud se Vám ukázka líbila, na našem webu si můžete zakoupit celou knihu.