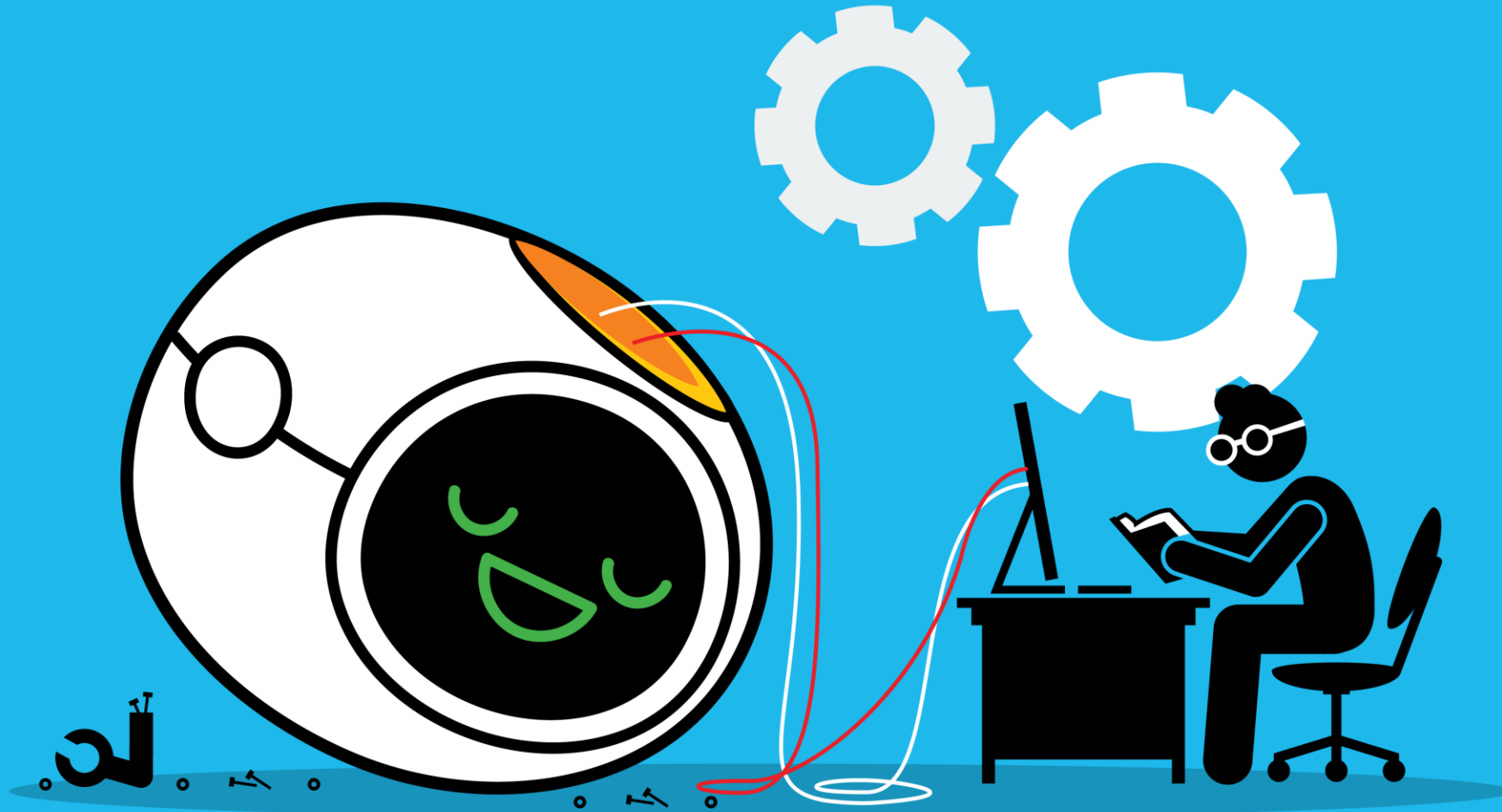




integrant

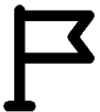
From Full Stack to Agentic: Solutions Engineering in the AI Era



Yousef Awad, CEO



● About Integrant



1992

FOUNDED AND
HEADQUARTERED IN
SAN DIEGO, CA



300+

ENGINEERS



Engineering team is
30%+ WOMEN



**INDUSTRY
EXPERTISE**

BIOTECH
MED DEVICE
MANUFACTURING
PUBLIC SECTOR
INFORMATION MANAGEMENT



**1 OUT OF
3 CUSTOMERS**

Have been with
us for 10+ years

**2 OUT OF 3
CUSTOMERS**

have been with us
for 5+ years



47% OF CUSTOMERS
have revenue over \$1B

24% OF CUSTOMERS
have revenue over \$7B

100% OF CUSTOMERS
WITH US HEADQUARTERS

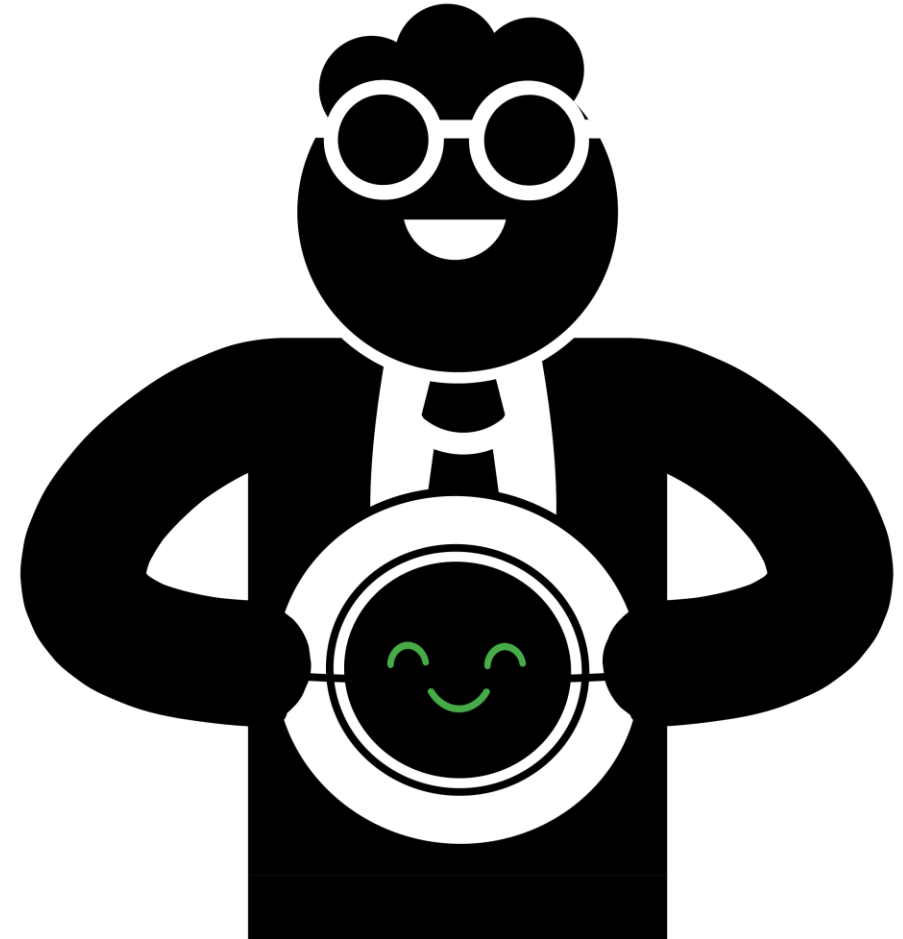
● Hype is High. Clarity is Low.



Every CIO I speak with is under pressure to do more than just 'experiment with AI.'

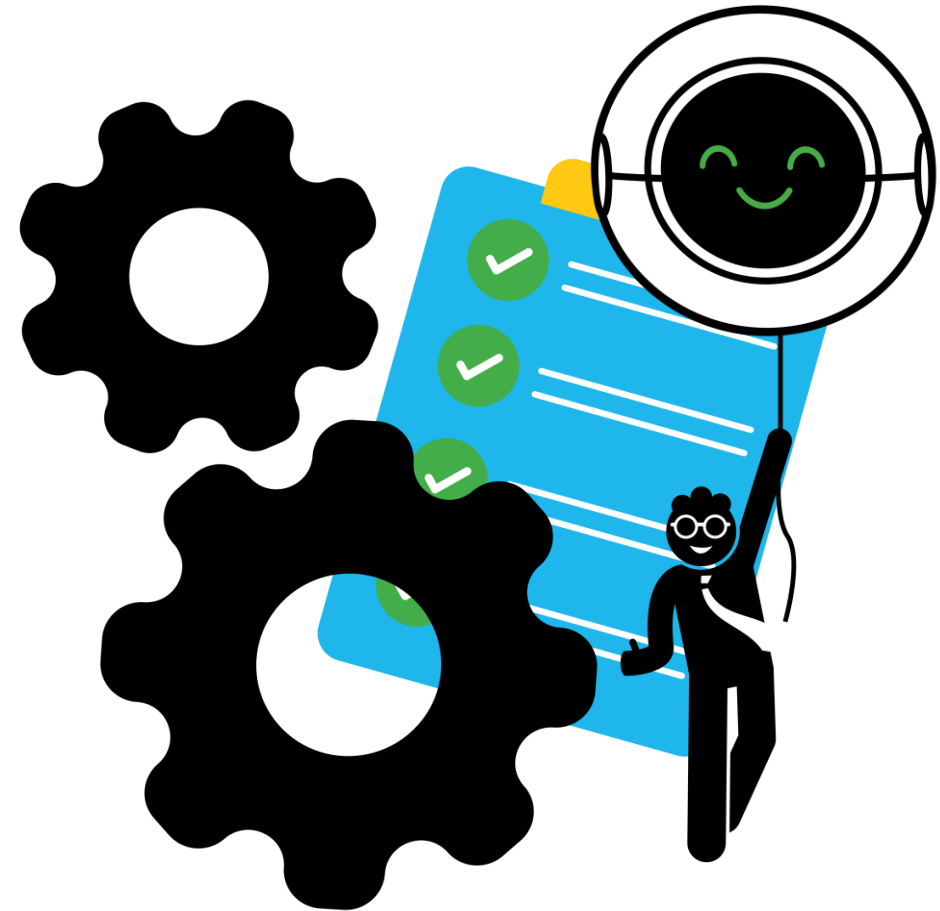
You're expected to **future-proof** your app portfolio, show **ROI** fast, stay **secure**, and somehow bring your teams along for the ride. Today we will talk about how to:

- Build AI-augmented applications and AI-ready teams
- Move the needle securely, responsibly, and with your engineers onboard



● Why You Can't Just Plug in Copilot or Cursor

- AI-generated code without architecture, security, or validation is risky
- Speed amplifies both strengths and weaknesses
- Enterprises need AI Software Engineers who can:
 - Evaluate AI-generated code for fitness and safety
 - Fit it into scalable, secure architectures



● AI is Ready. Are Your Teams?



- It's not just about tools, it's about:
 - Mindset shift
 - Responsible engineering and solid fundamentals
 - Continuous learning – not just a skillset upgrade
 - Team structure changes

Starting State	Transitioning To
SDLC roles (fullstack engineers, quality engineers, DevOps, data, data science, etc) using GenAI tools such as Cursor, CoPilot, etc.	AI Solution Engineers fluent in prompt engineering, system design, security, domain knowledge, cost optimization, and agentic tools

● What Makes a Developer AI-Ready



Integrant SME: “AI is powerful, but your real strength as a developer is in your brain—not just your tools. Don’t let your coding skills go soft.”

Prerequisites to Build Responsibly with AI

- System architecture foundations
- Expertise in design patterns
- Modular codebase skills
- Prompt engineering mastery
- Secure coding practices
- Scalability and maintainability mindset
- Testing and validation proficiency
- Growth mindset
- Domain knowledge

AI tools don’t replace skills; they amplify them with the right mindset and foundation.

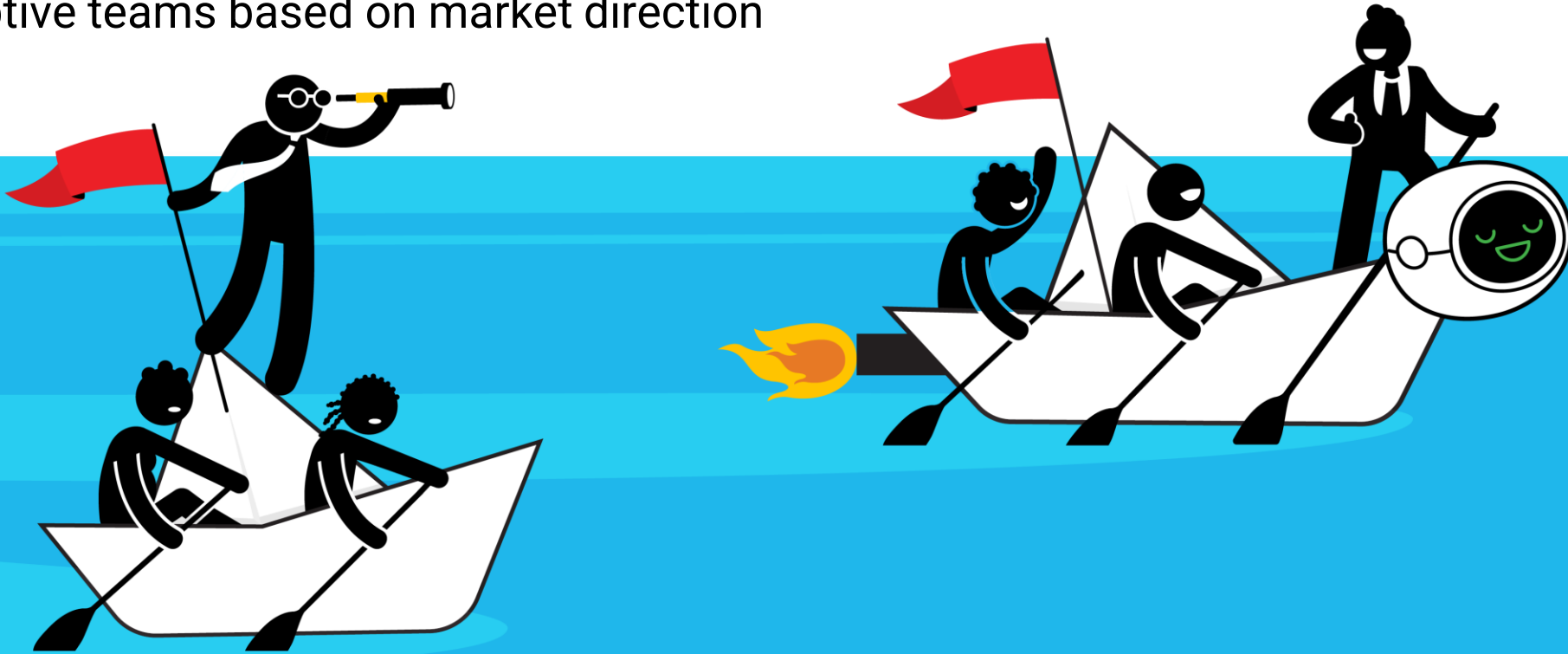
● Why AI-Ready Teams Must Be Domain Experts

- Accurate validation of AI outputs requires context
- AI amplifies good decisions and bad ones
- Teams organized by domain accelerate impact



● Preparing for Flexibility

- Developers trained across multiple stacks (e.g., .NET, Java, Python, JS, etc)
- Multi-skilling - Data engineering and DevOps exposure
- Adaptive teams based on market direction



● Agents We Built to Improve SDLC



Security Review Agent

- Scans source code, config files, and IaC for security vulnerabilities
- Integrates with DLP tools to prevent sensitive code from being used in AI generation
- Flags risky changes before they hit production
- Outputs auto-logged in Jira/Azure DevOps

Code Quality Agent

- Uses LLMs + domain knowledge to review logic, structure, and readability
- Catches anti-patterns and recommends refactoring
- Custom-trained on architecture docs, guidelines, and historical feedback



● Agents We Built to Improve SDLC

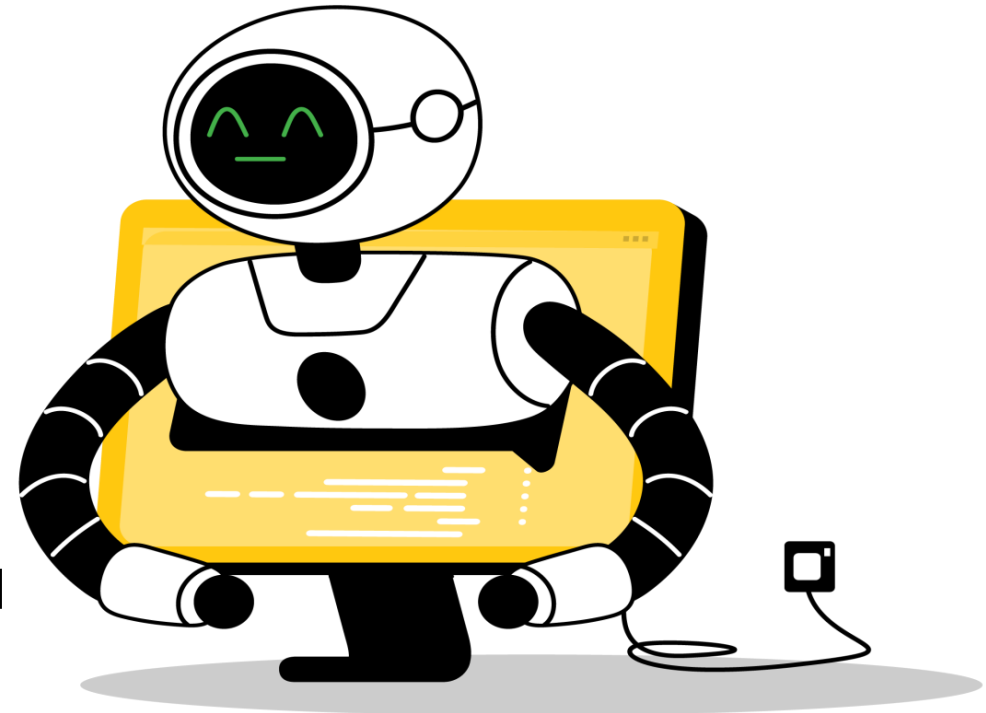


Knowledge Agent

- Builds a private, RAG-enabled knowledge base from architecture docs, PRs, RFCs, and FAQs
- Developers can query contextually to resolve blockers faster
- Helps onboard new engineers with instant access to tribal knowledge

Agent Orchestrator

- Coordinates workflows between other agents
- Uses A2A communication to define shared goals and review scope
- Example: Code Quality Agent + Security Agent collaborate on a secure code review pipeline



● New Challenges, New Mindsets



Engineers: Move from traditional application design to cost effective problem-solving using agents (with domain expertise and user acceptance mindset)

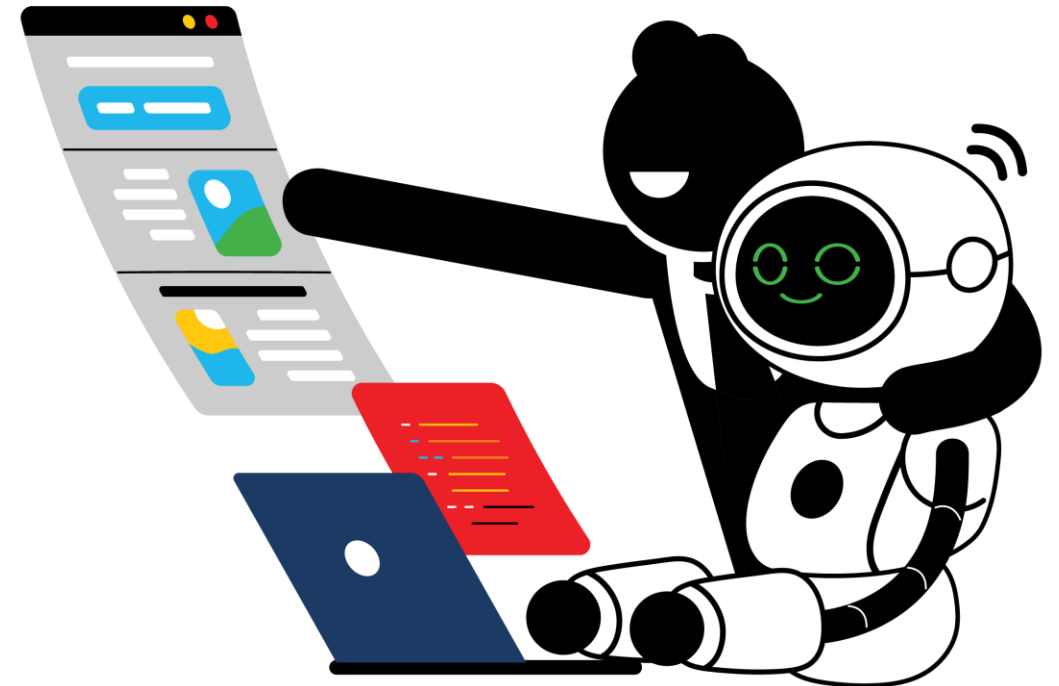
Customers: Move from “we have analytics” to “this is deeper insight”

- e.g., Predicting vendor reliability pre-signature or “what to do next” recommendations
- Not just analytics, but decisions made smarter and faster



● From Applications to Agents: The Shift to Agentic AI

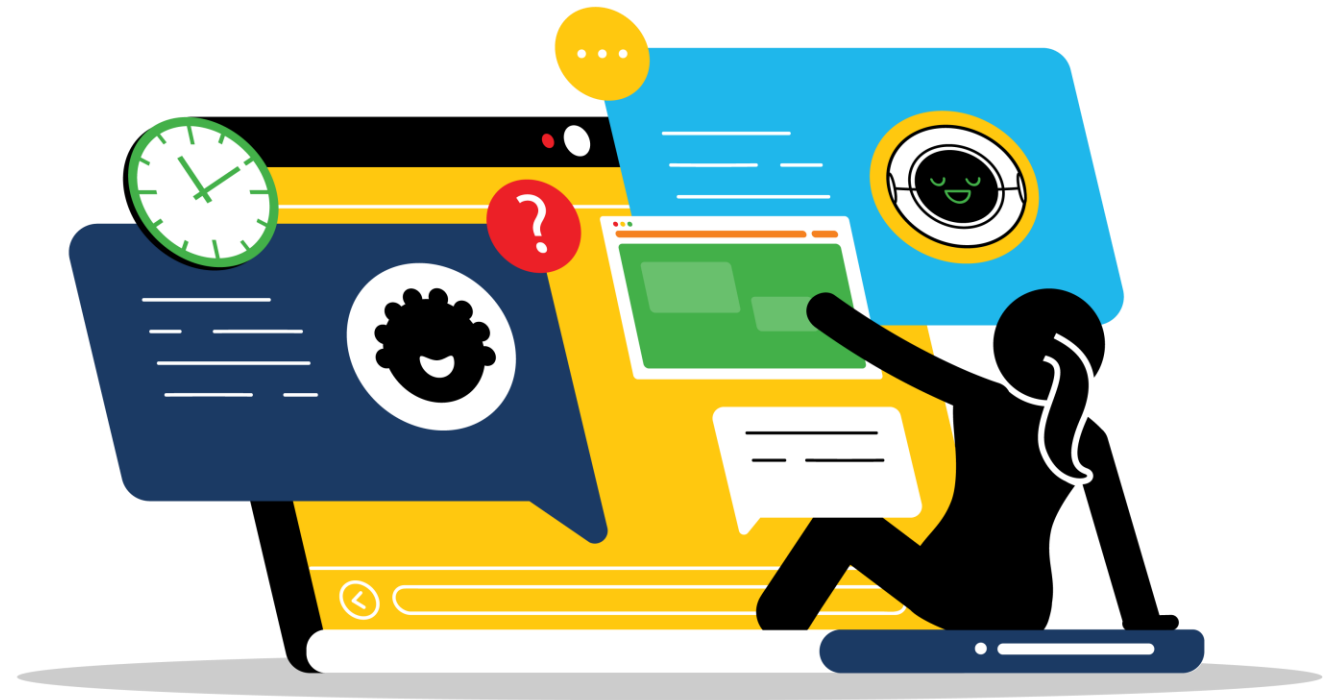
- **Personal agents:** automate small, repetitive tasks
- **Enterprise agents:** collaborate, reason, act
- Building **enterprise agents** requires:
 - MCP (multi-component protocols)
 - Agent-2-Agent communication (A2A)
 - Tools like RAG, LangChain, Vector DBs



● POCs Before Products

Start small, test fast, fail faster

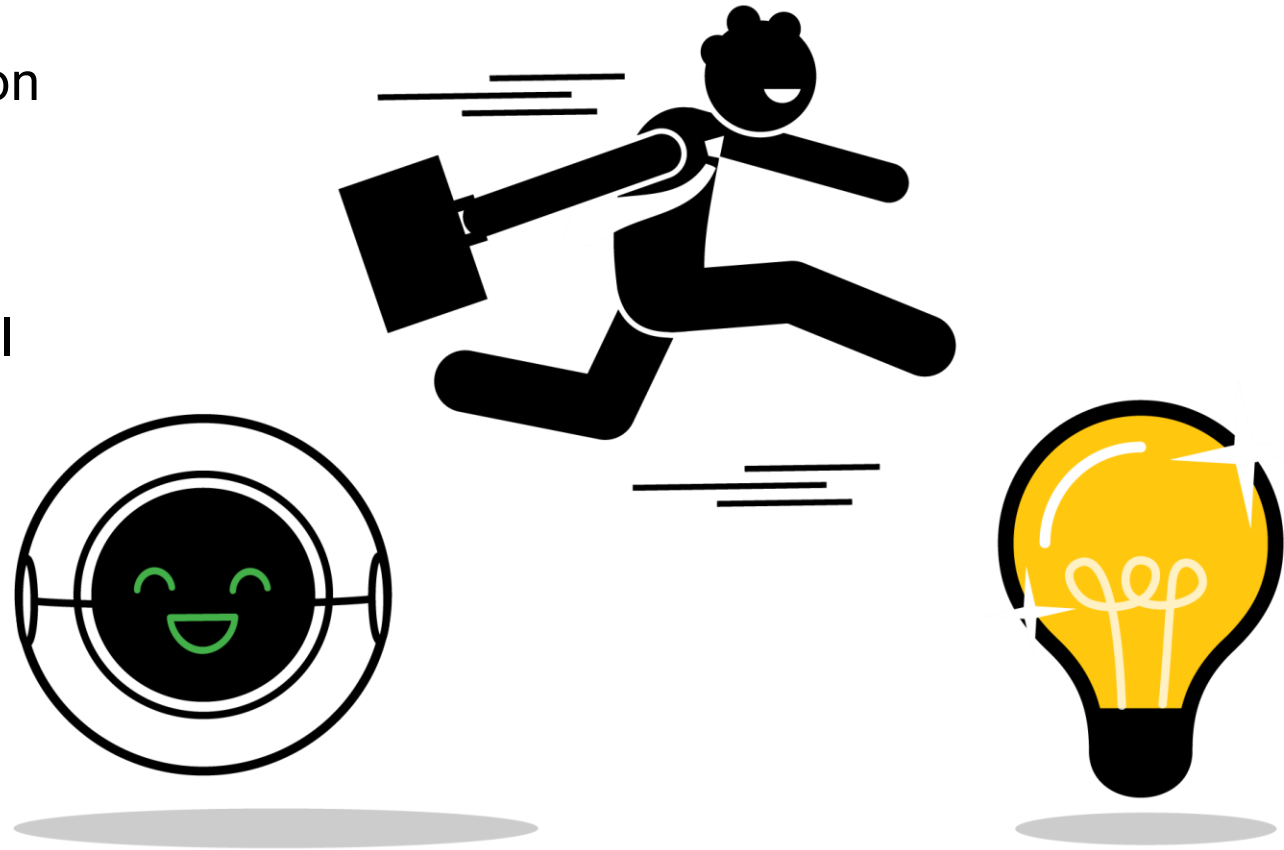
- Tools like Cursor, Windsurf, N8N, and Copilot help validate ideas or allow individuals to create usable solutions
- Once proven, move to full agentic architecture



● Engineering Culture for the Agentic Era



- AI doesn't replace strong collaboration or communication
- AI doesn't remove business alignment, it demands more
- Developers must evolve into AI Solution professionals



● AI Solutions Engineer Progression

Level 1: AI Integration Engineer - *AI-enhanced apps using open tools securely*

- Understand LLMs & embeddings
- Build simple MCPs
- Use open-source LLMs (e.g., Ollama, Mistral)
- Automate workflows (n8n, Airflow)
- Implement RAG with vector DBs

Level 2: AI Agent Engineer - *Complex, reliable agents that act autonomously*

- Build intelligent agents (Langchain, Autogen, CrewAI)
- Chain tools, APIs, and internal systems
- Add cost awareness and cloud-native skills

Level 3: AI Solutions Architect - *Secure, scalable AI systems with business impact*

- Deploy LLMs at scale
- Optimize cost, latency, accuracy
- Design for security & compliance

● **Beyond Hype: Strategic, Secure, Scalable**

It takes time to transform talented, professional engineers into AI Solution Engineers

Set up a team of your top engineers and subject matter experts to stay on top of the changes – the rate of change is accelerating

Start building or leveraging agents for IT – the SDLC, Service Desk, DevOps, and more

Teams need structure, guidance, and safe experimentation space



● Let's Talk



- What's holding you back from AI adoption?
- Curious about agents, RAG, or team setup?



Yousef contact

Yousef Awad

CEO

 /yousefawad

THANK YOU

CONTACT US FOR A CONSULTATION TODAY

Phone: + 1 858-731-8700

Address: 16870 W Bernardo Dr, Suite 250 San Diego, CA 92127

Web: www.integrant.com

● Beyond Hype: Strategic, Secure, Scalable

It takes time to transform your engineers into AI Solution Engineers
You need people who are constantly on top of what's new
Teams need structure, guidance, and safe experimentation space

AI agents are here

Teams need structure, guidance, and safe experimentation space

Tools are powerful, but the skills are critical

The world is changing rapidly, and we need teams to be flexible and ready to adapt



● Where It's Working



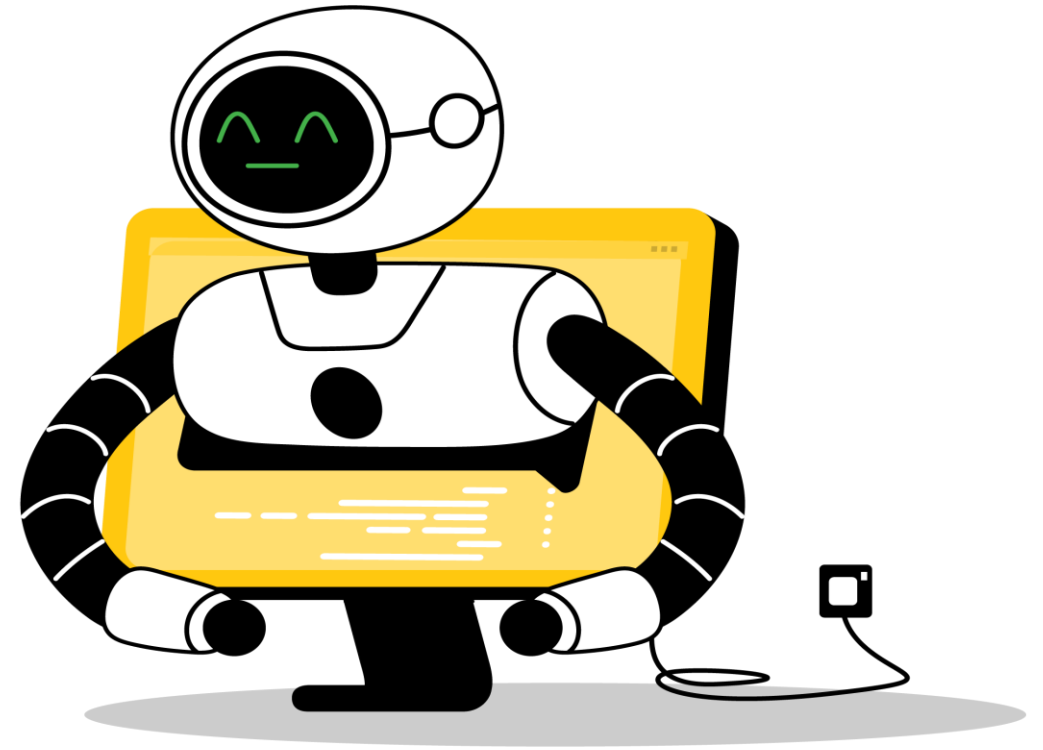
- Customer A (Parchment)
- Customer B (AAM)
- Customer C (CCBCC)

● Agentic Architecture Requires New Skills



Beyond Code:

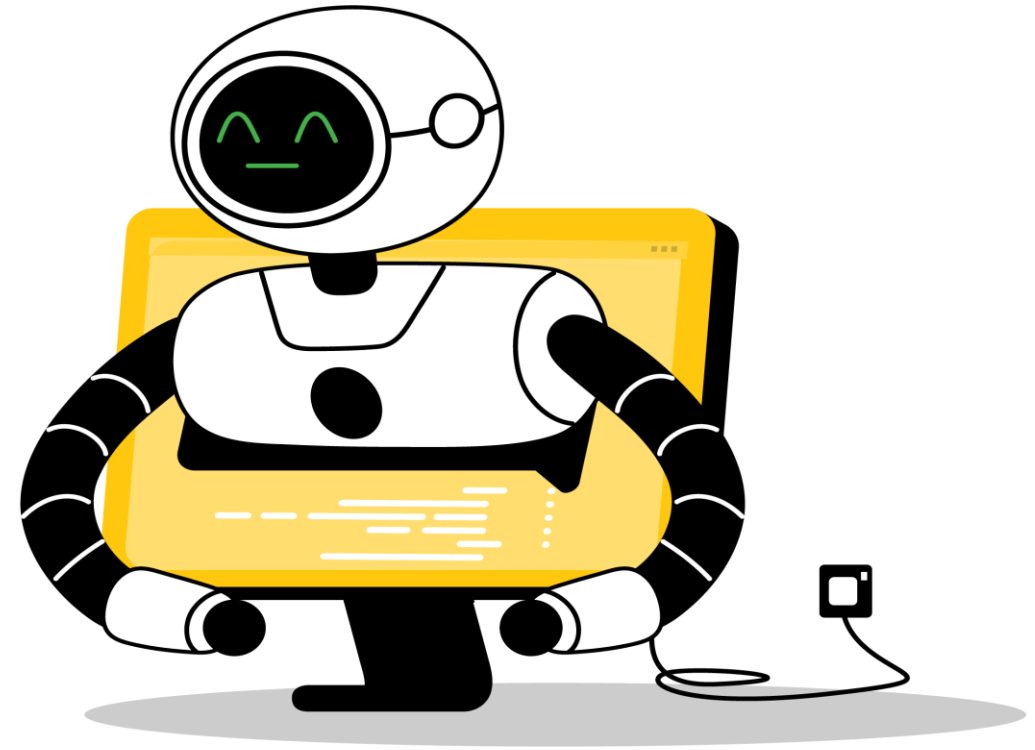
- Deployment across cloud environments
- Monitoring real-time LLM output quality
- Prompt testing at scale (100+ prompts)
- Continuous improvement via feedback loops



● AI Agents

Web Chat Agents or Background Worker Agents

- Reason and Answer questions based on world class LLMs
- Use Tools to access the internet
- Use Vector Database to access indexed internal documents
- Use Database or API access to internal systems to retrieve data or take actions on behalf of the users
- Fully controlled access to internal systems, user/role-based access
- Use MCP to allow agent access tools like the Internet, Vector DB, Database or API for internal systems
 - Use A2A (Agent 2 Agent) to allow agent to agent communication

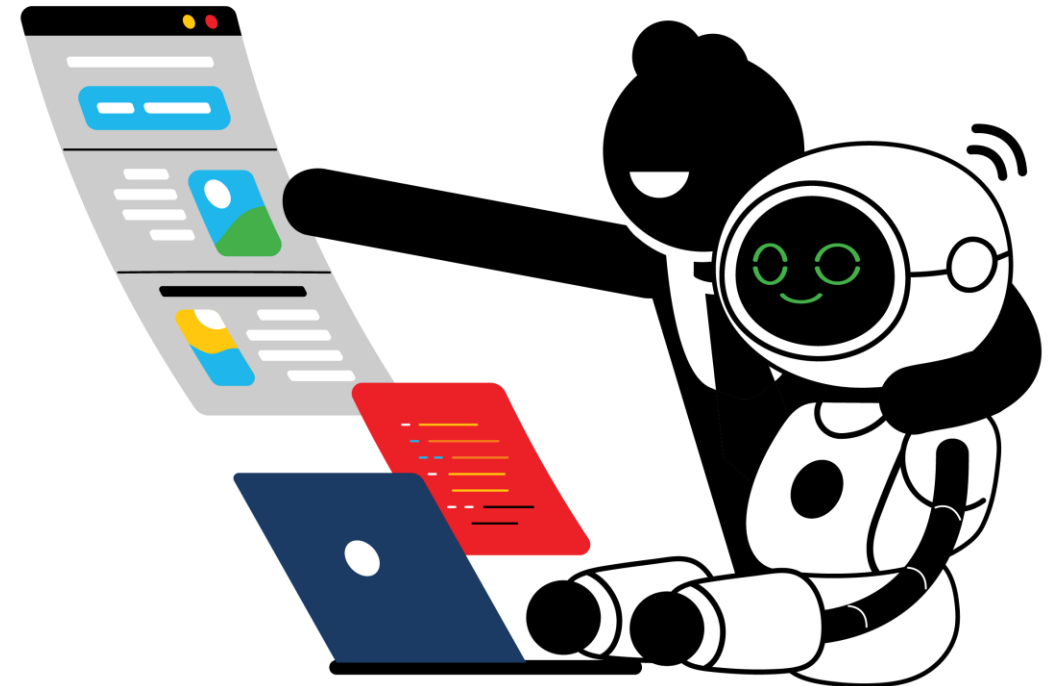


● Beyond Fullstack: Becoming AI Engineers



- Prompt engineering
- RAG frameworks
- Secure tool stack (Cursor, Supabase, Vercel)
- Architecture + system thinking
- Integration with LLM APIs (e.g., OpenAI)
- Use of vector databases, RAG, orchestration
- Designing for agent-based windows

- Agents > Apps
- Learning > Languages
- Domain + delivery = speed

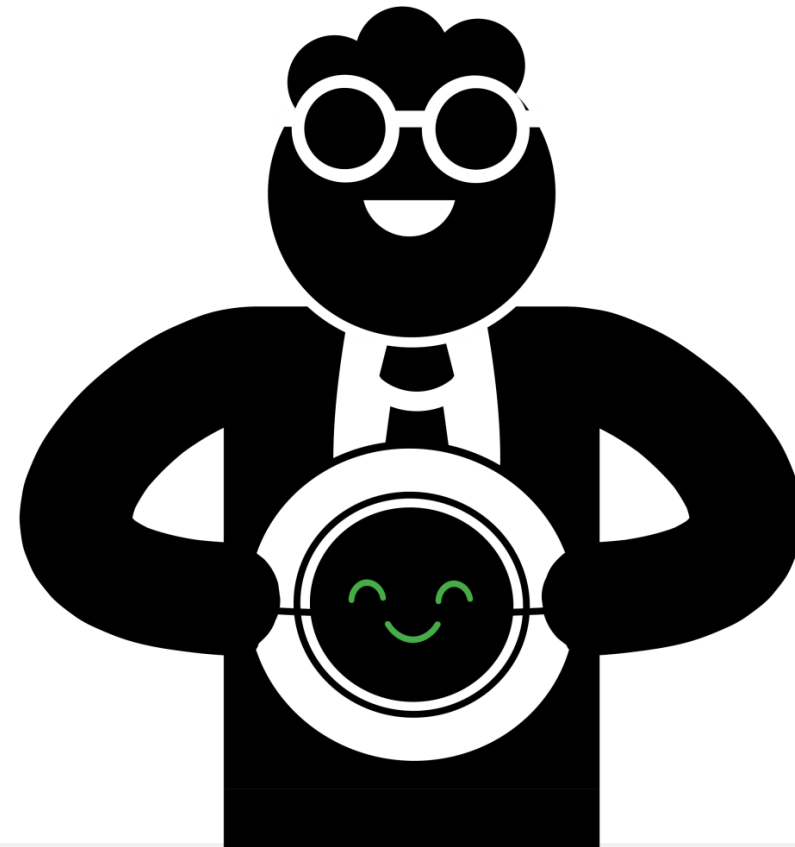


● Skills & Tools We Prioritized



- Prompt engineering
- RAG frameworks
- Secure tool stack (Cursor, Supabase, Vercel)
- Architecture + system thinking

- Agents > Apps
- Learning > Languages
- Domain + delivery = speed



● How We Evolved Our Engineering Org



- 300+ engineers. Originally:
 - X% web
 - X% mobile
 - X% etc
 - Moved to X% multi-lingual, test automation = developers, etc etc
- R&D Teams / LaunchPad
- AI-focused L&D roadmap / tech sessions
- Tool access
- Cross-stack fluency via vibe coding
- Hiring for curiosity, innovation, and problem solving, not tech stacks



● Who Might Benefit – Optional












- CIOs and tech leaders exploring how to prepare their engineering orgs for AI
- Teams early in their AI journey
- Those looking for a practical roadmap







● From Developers to Developers + Agents



Traditional team structure (example)

- 1 TPL 
- 4 Developers    
- 2 SDET  
- 1 DevOps 
- 1 Data Dev 

New team structure

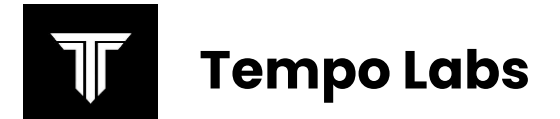
- PO 
- 2 dev leads  
- Agents 

Agents augment teams and automate workflows

● Proposed Tech Stack



Purpose	Current Stack	Proposed Stack for Dev Env	Proposed Stack for Production
Design/Dev	Visual Studio	TempoLabs + Cursor	Cursor
Frontend	Angular	NextJS (React)	NextJS (React)
Backend	.NET	NextJS	NextJS
Database	Azure SQL	Supabase	Azure Database for PostgreSQL
Application Hosting	Azure App Service	Vercel	Azure App Service
Authentication	Akamai	Supabase	Custom implementation for Akamai



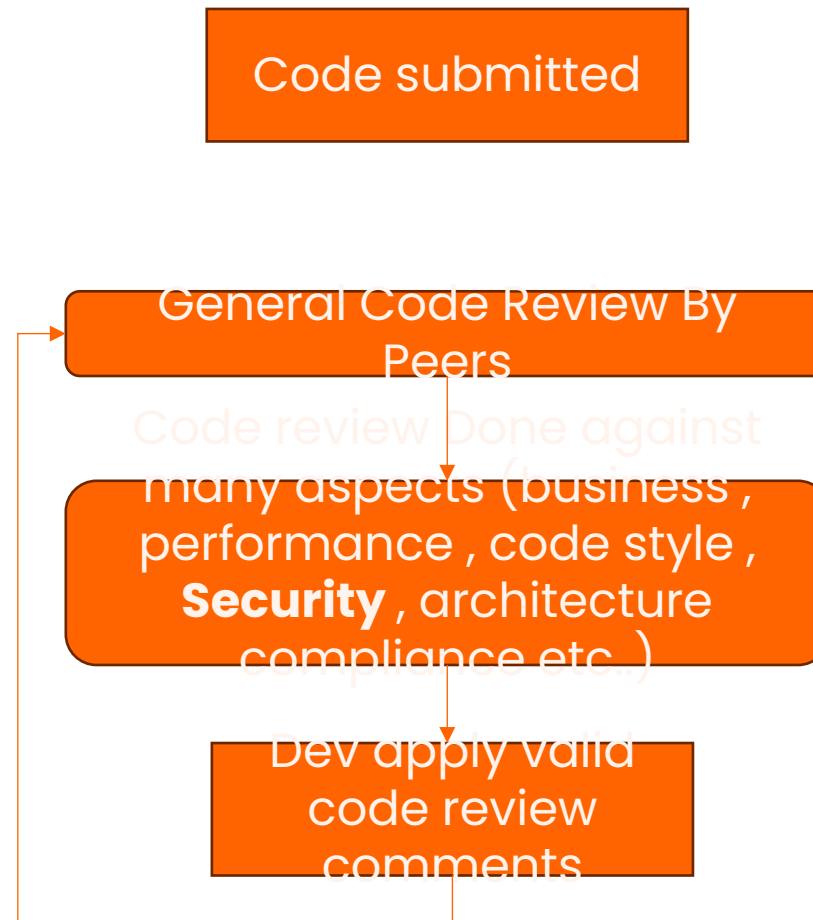
● Five Steps for CIOs

1. Assess your team's AI fluency
2. Train before you tool
3. Test ideas with low-code agent tools
4. Communicate the transformation
5. Rethink team structure with agents in mind





Traditional Process



Challenges :

- *manual review can bypass issues specially with large code changes
- *Developer are reviewing code against many aspects at the same time
- *No focus on changes on production configuration files from security perspectives

Meet 'CyberMorph'



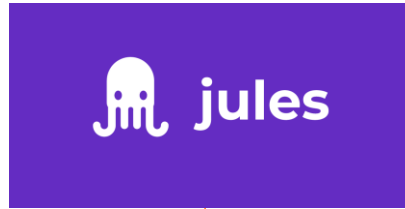
● Integrant AI Security Agent



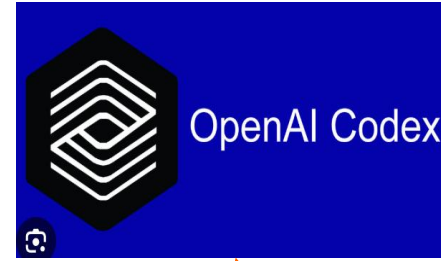
Static code analysis tool



Software Eng Agent 1



Software Eng Agent 2



Software Agent 3
FIM/XDR Agents

wazuh.



Azure Sentinel



Azure Security Center



What we want to achieve – By CyberMorph



Static code analysis tool



Software Eng



Software Eng Agent 2



Software Agent 3
FIM/XDR Agents

wazuh.



Azure Sentinel



Azure Security Center



CyberMorph- in a nutshell



security Layers

1. Static code analysis – e.g sonarqube
2. Advanced analysis using software eng Agent Codex/Jules
3. Production environment configuration Monitoring FIM/XDR integration

Unauthorized AI tools generation prevention

1. DLP integration currently cyberMorph can stop AI tools generation for C# and Java files for backend modules , yet allow access for other non client related projects. This is done by pattern matching for main application namespace and packages.
2. DLP integration for frontend code is still challenging , an idea related to folder structure matching against github folder structure in mind to control access (brainstorming)

Morph final output

1. Code review and security issues will be created and reported as bugs automatically on Agile boards like Jira/Azure Devops
2. Alerts/reports will be sent automatically

CyberMorph- Jules simplified workflow

1-Setup and Rag



Provide source control access and additional documents that can be added to context like architecture documents , knowns issues etc..

CyberMorph- Jules simplified workflow

2-Agree on code review plan



The 2 agents will discuss the scope of the code review and infer focus on security issues

can you check the full code base?

Jules is waiting for your input to continue working

Reviewing the entire codebase is a significant undertaking. To make this feasible, could you indicate your preferred approach? For example, should I focus on critical modules, use automated checks with manual spot-checks, review a sample of the code, or conduct a time-boxed review?

do the critical modules

Great. Could you help me identify the critical modules? For example, what are the core functionalities, areas with recent significant changes, modules handling sensitive data, or parts essential for stability/performance?

focus on stability and security

Okay, focusing on stability and security. Based on that, I've identified some key areas like authentication, API endpoints, database interactions, core business logic,

CyberMorph- Jules simplified workflow

3-Start review based on agreed plan



The 2 agents agreed on review scope/steps then Jules start the actual code review

Jules is waiting for your input to continue working

- 1 Review Authentication and Authorization (app/api/auth/)
- 2 Review API Endpoints and Command Handlers (app/api/.../route.js, app/api/command-handler/)
- 3 Review Database Interaction Logic (lib/db/ and related usage)
- 4 Review Core Business Logic in Key Features (lib/features/)
- 5 Review External Service Integrations (lib/integrations/)
- 6 Review Socket/Real-time Communication (lib/socket/, socket-server/)
- 7 Summarize Findings and Provide Recommendations

Hide

CyberMorph- Jules simplified workflow

4-final step of review and logging



Jules perform code review and share results with cyberMorph who will log new issues on the Agile board like azuredevops and Jira

Reviewed NextAuth configuration.

Key recommendations include ensuring secure management of environment variables (especially NEXTAUTH_SECRET) and verifying redirect URIs in provider consoles. Authorization logic will be reviewed in subsequent steps.

Reviewed command-handler and an example 'add method' API endpoint.

Key recommendations: ensure specific validation in dynamic handlers (like command-handler's sub-handlers), implement robust authorization checks for all API actions, improve error response security by not leaking raw error details, and use database constraints to prevent race conditions in create/update operations. Use of Zod for input validation is a good practice.

Reviewed Cosmos DB interaction logic.

Critical NoSQL injection vulnerabilities found due to query string interpolation; these must be fixed by using parameterized queries. Recommendations also include improving ID generation to avoid bottlenecks/race conditions, refining error handling, and carefully evaluating utility functions like `createFromCSV` and `deleteAll` for production use. Some queries correctly use parameterized queries, which is good.