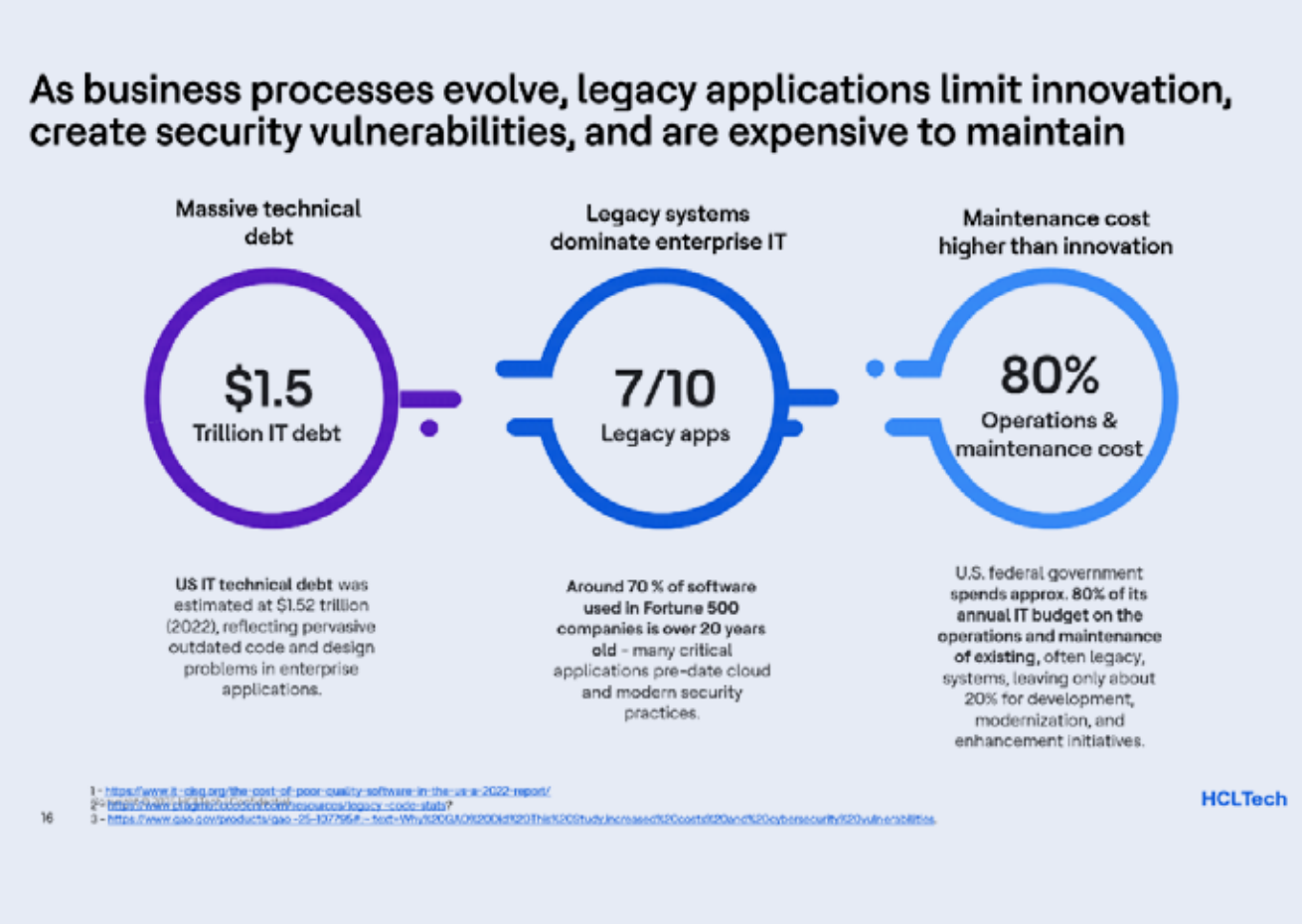
HCLTech | OpenAI

Accelerating application modernization with OpenAI Codex

A practical approach to code understanding, refactoring, testing and migration with human oversight

Executive summary

Large enterprises today face immense pressure to modernize their legacy applications to stay competitive, reduce security and operational risks. Decades-old systems, from COBOL mainframes to aging Java or .NET monoliths, underpin critical business processes in almost all industries and increasingly impede innovation by slowing down product development, exposing security vulnerabilities and making integration with modern tools difficult. In 2022, just in US, the IT technical debt was estimated at $1.52 trillion, reflecting pervasive outdated code and design problems in enterprise applications. Around 70 % of software used in Fortune 500 companies is over 20 years old, many critical applications pre-date cloud and modern security practices. U.S. federal government spends approx. 80% of its annual IT budget on the operations and maintenance of existing, often legacy, systems, leaving only about 20% for development, modernization and enhancement initiatives. To better serve customers, enterprises seek improved performance, better integration capabilities, enhanced security and agility to support new digital initiatives.



Traditional modernization efforts are fraught with challenges. Many migration and modernization projects suffer cost overruns, missed timelines, or even outright failure. Industry research indicates that up to 68% of legacy modernization projects fail or fall short of expectations. Common failure modes include multi-year projects stalling without ROI, budgets that spiral due to unforeseen complexity, or "modernized" systems that still carry old technical debt under the hood. The core issues are often not just technical but stem from limited system understanding and knowledge gaps, hidden dependencies, undocumented business rules and lost expertise as veteran developers retire over time.

AI-assisted development is emerging as a critical enabler to break through these hurdles. Recent advances in GenAI, exemplified by OpenAI's Codex models, have shown promise in accelerating application migration and modernization tasks. OpenAI Codex offers a transformative assistive capability that can significantly boost productivity by accessing the application code, developing an extensive documentation, determining appropriate modernization approach for each application (R type), converting legacy databases to modern cloud-based data lakes or warehouse, code development, implementing CI/CD pipelines to automate testing and deployment or even redesigning the UI for better user engagement and many more. By pairing human expertise with AI's speed and pattern recognition, organizations can overcome legacy challenges, save time and achieve meaningful modernization of their application estate to meet the demands of the digital age.

Challenges in application migration & modernization

Enterprises face key challenges when attempting to migrate or modernize legacy applications using traditional approaches:

- **Legacy codebase and complexity**: Legacy applications often consist of millions of lines of code accumulated over decades, written in outdated languages (COBOL, PL/1, VB6, older Java/.NET frameworks, etc.) and tight coupling. These systems have complex interdependencies and "spaghetti code" structures that are difficult to understand. For example, in banking, as of 2025 an estimated 220 billion lines of COBOL are running in production worldwide, powering core ledgers and batch processes. Furthermore, many of these core applications haven't been significantly updated in 20+ years. The sheer size and antiquated structure of such codebases make manual analysis extremely time-consuming which makes development teams struggle to understand what each part of the codebase does and how changes will ripple through, eventually impacting modernization timelines and efforts.
- **Knowledge and documentation gaps:** Many systems were built in the past show a "fog of uncertainty" about system internals and workings. As veteran engineers retire or leave, their expertise walks out the door, creating a knowledge vacuum. For instance, an average COBOL programmer is nearly 58 years old and about 10% retire each year, steadily shrinking the pool of experts. The result is that many legacy systems lack adequate documentation and skilled support, forcing modernization teams to reverse-engineer code and reconstruct understanding from scratch. A 2025 industry survey noted that maintaining legacy systems is often compounded by limited documentation and shrinking institutional knowledge due to staff turnover. This knowledge gap increases the risk of errors and slows down any migration project.
- **Cost, risk and timeline overruns:** Unexpected complexities, such as undocumented dependencies, data quality issues, or scope creep, frequently surface mid-stream and derail plans. One study found that 68% of legacy modernization initiatives either fail outright or fall short of their goals. These overruns stem from underestimating legacy complexity and over-reliance on a few subject matter experts. The risk is compounded by the mission-critical nature of many legacy apps, downtime or defects introduced during migration, cause serious business disruptions, further extending timelines.
- **Technology constraints and external pressures:** Legacy systems often run on antiquated hardware or operating systems, use databases or languages no longer supported and lack compatibility with cloud or API ecosystems. This makes extracting data or integrating new features difficult without extensive workarounds. Also, external pressures like cybersecurity threats, regulatory compliance demands and the need to leverage data for AI/analytics make migration further complex. Many organizations recognize that legacy software isn't just outdated, it's actively getting in the way of innovation. Legacy platforms trap data in siloed formats and make it "nearly impossible" to feed modern AI or machine learning tools. This makes modernization hard, technically challenging, resource intensive and easy to defer.

Given these challenges, AI-assisted approaches like OpenAI Codex are gaining attention, offering approaches for tackling these obstacles in a more efficient and controlled manner.

OpenAI Codex

At its core, Codex is OpenAI's series of AI coding tools that help developers move faster by delegating tasks to powerful cloud and local coding agents. In the context of legacy system modernization, Codex can help to faster understanding of application codebase, quicker implementation of new code and fewer manual steps for the development team. Codex can assist at various stages of modernization cycle, from analyzing a monolithic codebase to transforming it into cloud-native microservices.

- **Legacy code understanding and summarization:** Codex excels at reading through large codebases and explaining them in plain language, can identify key components, dependencies between modules and describe system functionality even if documentation is missing. This is especially valuable when original developers are no longer available, or applications have missing documentation. Codex provides clear explanations of legacy code behavior and even generating technical documentation like user stories, design diagrams and data model descriptions from analyzing the code. By automating code comprehension, Codex enables modernization teams to gain confidence in understanding current state functionality much faster, which is crucial before attempting to change or refactor anything.

- **Automated refactoring (monolith to microservices):** A common modernization pattern is breaking a monolithic application into microservices or modular components. Traditionally, this requires architects to manually determine service boundaries, extract code and rewrite interfaces, which is a time-consuming process. Codex can significantly accelerate refactoring by proposing and even implementing code restructuring. Codex can analyze the monolith's codebase, suggest how to slice it, create new functions or classes and update references, all while aiming to preserve original behavior. This dramatically reduces the effort to transform a monolith's internal structure or extract microservices, which otherwise is one of the most labor-intensive parts of modernization.

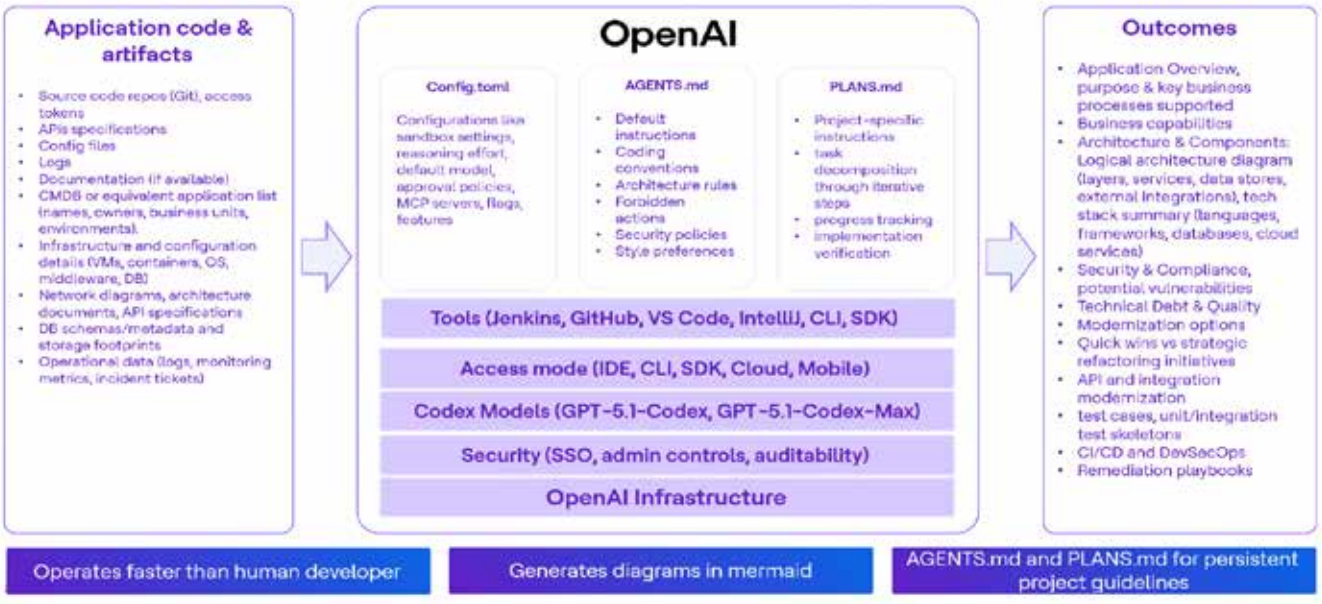
- **Language transformation:** Porting an application from a legacy programming language to a modern one is another scenario where Codex offers significant value. It can act as an intelligent code translator, converting code from one language to another while understanding the intent behind it. For example, Codex can convert COBOL to Java, recognizes COBOL file I/O and suggests equivalent JDBC database operations, or replaces procedural logic with object-oriented structures. Codex is familiar with historical coding patterns and can suggest modern equivalents while preserving functionality. This language transformation capability can jump-start efforts to retire outdated technology.

- **API enablement:** One of the modernization approaches is to expose legacy functionality as APIs or microservices so that it can be reused and extended. Codex can help identify logical service candidates within a legacy codebase. For example, by analyzing a large system, Codex might highlight that "these functions relate to payment processing, which could be an API" versus "these parts handle reporting." Also, Codex can generate the wrapper code to expose a legacy component as a service. In case of wrapping a COBOL program as a REST API, Codex can generate a REST API scaffold (in Java or Node.js) that calls the COBOL program and returns the result as JSON. Codex can produce the API handler code, data transformation logic and even documentation for the new service. This level of automation drastically reduces the manual effort to API-enable legacy systems. Moreover, because Codex can generate OpenAPI/Swagger documentation, it can document the new service contract as it creates it. By extracting self-contained services from the monolith to API, enterprises can start operating in a more agile, decoupled fashion and create necessary integration code on demand.

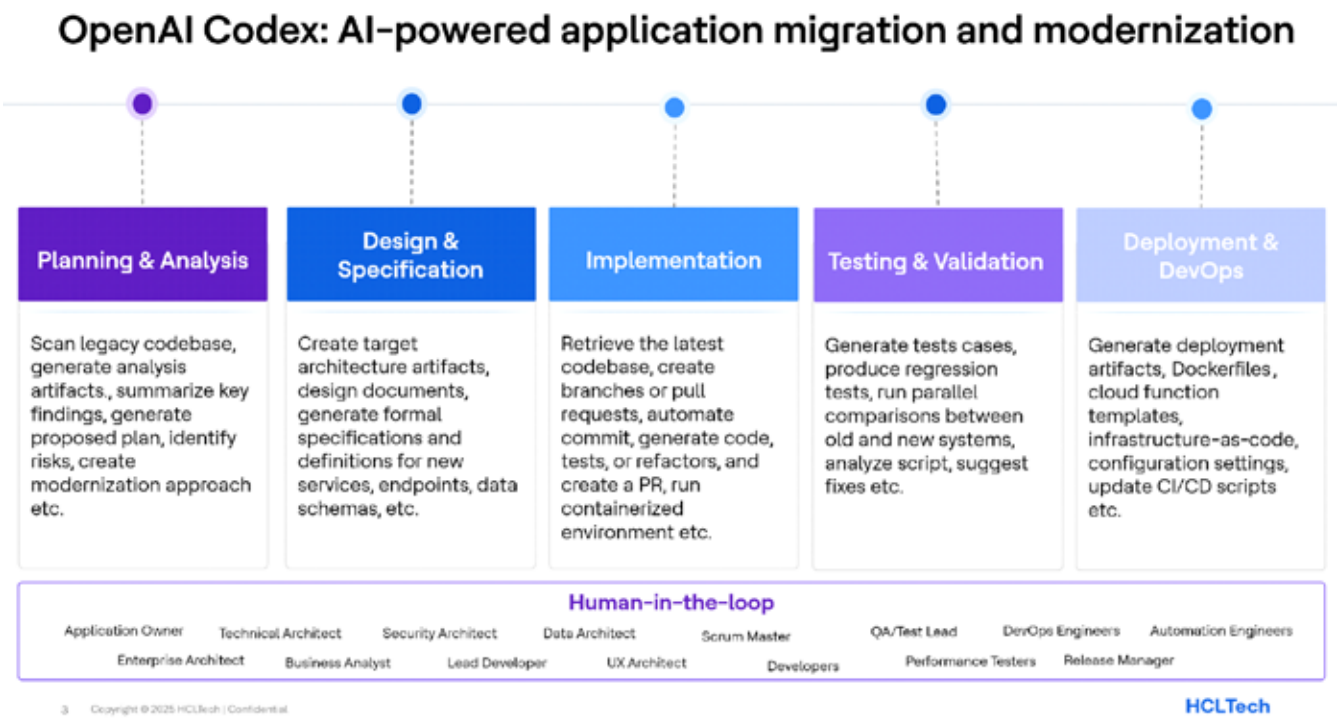
- **Test case generation and automation:** Codex can aid QA teams by generating test cases and test code automatically. It can infer what output is expected for certain inputs, identify edge cases (such as null or boundary values) and create assertions accordingly. This is incredibly valuable for legacy systems that often have poor test coverage. Codex effectively helps backfill tests to ensure that behavior is preserved during modernization. Additionally, Codex's capabilities go beyond unit tests as it can outline integration test scenarios or even help set up parallel run comparisons. One strategy is to run the old and new systems side by side on the same inputs and compare outputs; Codex can assist by creating harness code that feeds test data to both versions and flags any differences. By automating test generation and providing a systematic way to verify functional parity, Codex reduces the risk that modernization introduces bugs, saves QA teams a tremendous amount of effort and makes testing cycle faster.

- **Cloud native migration:** Codex can facilitate cloud migrations by generating the boilerplate code and configuration needed for cloud environments. For instance, if containerizing a legacy application, Codex can write a Dockerfile or Kubernetes YAML manifest. Similarly, for serverless migrations, Codex can port a function into an AWS Lambda handler format or an Azure Function template. Codex has knowledge of common cloud SDKs and configuration formats, so it can produce not only application code but also the DevOps scaffolding around it. Codex can ensure that details like environment variables, container build steps, or cloud function signatures are correctly handled and follow the best practices. This helps teams move legacy workloads to modern platforms faster and with fewer errors in configuration and expediting migrations.

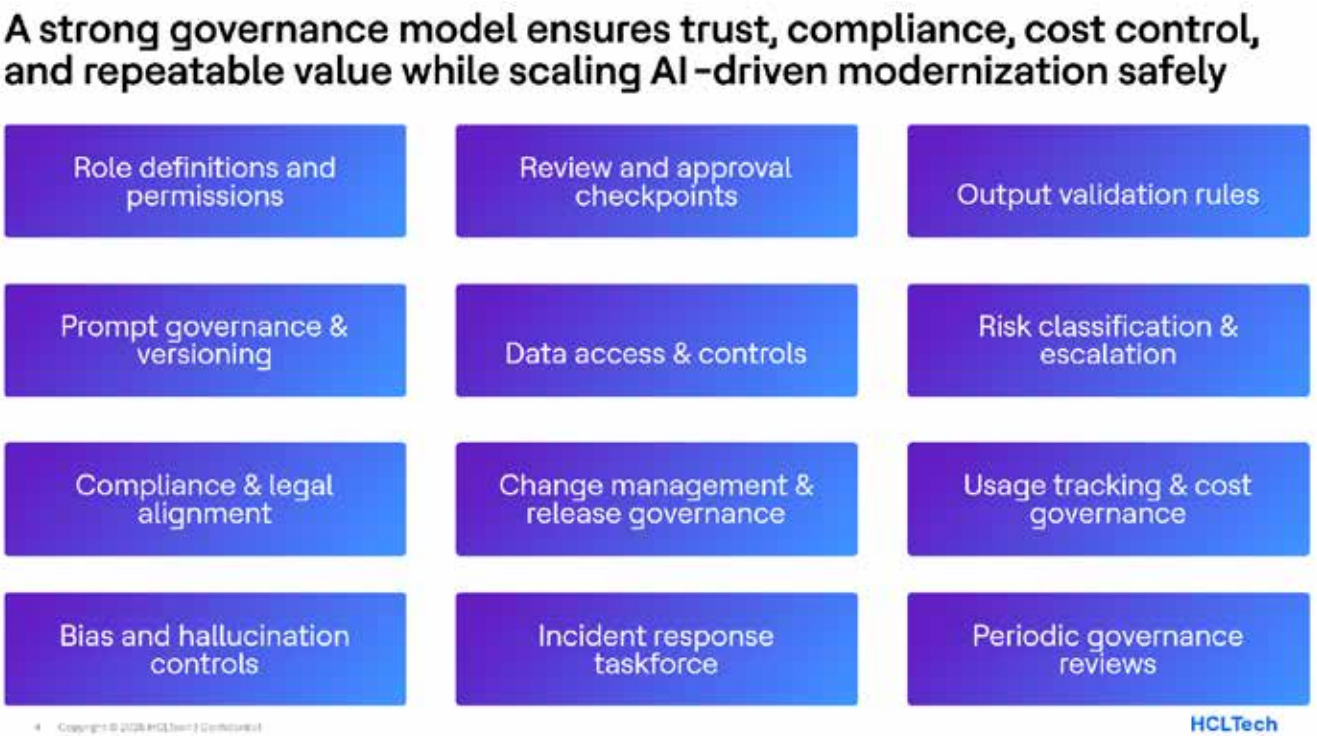
Application migration and modernization through OpenAI Codex



To fully realize the benefits of Codex in enterprise modernization, enterprises should integrate it into their Software Development Life Cycle (SDLC) and toolchains.



While Codex is powerful, human oversight is imperative to ensure reliability, security and alignment with business goals. A governance model for AI-assisted coding in modernization is required so that Codex driven automations are plan-driven, reviewed and validated by respective teams.



Business benefits and value realization

Adopting OpenAI Codex in application migration and modernization efforts can yield substantial business benefits. While the use of Codex and similar platforms is picking up the momentum, few of the businesses have observed significant benefits practice, from pilot implementations to case studies, underscoring how Codex accelerates modernization efforts while improving outcomes across cost, quality and time-to-value. The result is a faster, safer migration journey and tangible business value realization in enterprise settings.

- **Reduced migration effort & timelines:** Automating code translation and refactoring with Codex drastically shortens migration projects. In practice, AI analysis eliminated ~30% of unnecessary code rewrite effort during a COBOL-to-Java migration, accelerating delivery.
- **Improved code quality & maintainability:** Consistent coding standards, thorough testing and up-to-date documentation in the modernized codebase leads to higher quality and easier maintenance for legacy-modernization projects. Organizations report stronger test coverage and cleaner, more maintainable code after AI-led migrations. AI-generated tests reduced post-migration downtime by ~40% through early issue detection.
- **Cost savings on development & testing:** Codex's automation shrinks the need for large developer teams to grind through legacy code, translating into 20–30% lower migration and maintenance expenses (fewer contractor hours, less rework). These efficiencies, combined with shorter project timelines, yield substantial budget savings while delivering improved outcomes.
- **Security & compliance risk mitigation:** Codex can automatically audit code for flaws or policy violations as it generates and refactors code. Proactive auditing and testing can shrink security remediation effort by an estimated 10–20% and ensures migrated systems meet compliance requirements from day one.
- **Accelerated time-to-market:** Faster migrations directly translate into quicker delivery of modernized applications and new features. By removing migration as a long-term bottleneck, organizations can roll out updates and innovations to customers much sooner, gaining a competitive edge.
- **Developers' productivity gains:** Real-world trials show developers completing tasks ~55% faster with AI assistance compared to existing coding methods. Teams using Codex-based tools have reported around 25% increase in overall development velocity.

Leveraging OpenAI Codex in modernization efforts provides a multi-faceted value proposition: speed, quality, risk reduction and cost efficiency. Enterprises can expect not only to save time and money on the technical migration itself, but also to accelerate their strategic outcomes by adopting new technologies, improving customer experience and responding faster to market changes.

OpenAI Codex in enterprise transformation: real-world use cases built by HCLTech team

- **Healthcare operations and patient experience improvements:** Healthcare providers face persistent challenges such as supply shortages, long patient wait times and strict regulatory constraints that limit the use of traditional video analytics. Using OpenAI Codex, teams rapidly developed privacy-preserving computer vision solutions capable of detecting and counting medical supplies from images, estimating patient wait times through waiting room analysis and automatically anonymizing sensitive data such as faces in video feeds. Codex enabled natural language-driven development and rapid prototyping across diverse hardware and deployment environments, significantly reducing the need for specialized expertise in vision systems. As a result, hospitals were able to improve inventory management, proactively manage patient flow and enhance safety outcomes while maintaining compliance with healthcare privacy regulations, ultimately reducing both operational inefficiencies and patient care risks.

- **Reverse engineering of large-scale legacy systems:** Organizations operating decades-old mainframe systems often struggle with undocumented code, accumulated technical debt and heavy reliance on scarce legacy expertise, making modernization efforts slow and risky. Leveraging OpenAI Codex, teams ingested large volumes of legacy code, including COBOL, PL/I and related artifacts and automatically reverse engineered business logic, extracted functional requirements, identified defects and reconstructed user personas from legacy interfaces. Codex further enabled the generation of modern architecture blueprints and design specifications through iterative, context-aware analysis. This approach reduced system comprehension timelines from months to hours, created a structured and reliable foundation for modernization and enabled downstream transformation initiatives such as API enablement and AI-driven automation, all while significantly lowering dependency on legacy subject matter experts.

- **Customer experience and operational optimization:** In the cruise line industry, delivering highly personalized guest experiences while managing complex operational constraints is critical for competitive differentiation. Using OpenAI Codex, teams rapidly built functional MVPs, including an AI-powered concierge capable of personalizing itineraries, supporting multimodal interactions (voice and chat), handling real-time service requests, and driving context-aware upsell opportunities. In parallel, an AI-based crew allocation optimizer was developed to manage scheduling by considering compliance requirements, cost constraints and workforce availability. Codex accelerated full-stack development through natural language-driven coding and rapid iteration, enabling tangible, working demonstrations. This resulted in faster stakeholder buy-in, shorter sales cycles, improved guest satisfaction, increased revenue opportunities and enhanced operational efficiency.

- **Agentic workflow automation beyond traditional RPA:** Enterprises often rely on traditional RPA solutions to automate repetitive workflows, but these systems tend to be rigid, costly to maintain and difficult to scale in dynamic environments. OpenAI Codex enabled a new approach by facilitating the creation of lightweight, agentic automation workflows that can execute browser-based tasks without requiring APIs, perform multi-step reasoning and adapt to changing interfaces. These AI-driven "skills" were used to automate activities such as product research across websites, comparative analysis and the generation of structured reports from unstructured data sources. Defined and orchestrated using natural language, these workflows significantly reduced development effort and increased flexibility. As a result, organizations were able to lower automation costs, accelerate deployment timelines, improve resilience to change and empower business users to automate processes without the need for complex engineering or heavy RPA infrastructure.

- **End-to-End mainframe modernization:** Legacy COBOL-based systems, while mission-critical, are increasingly costly to maintain and lack modern capabilities such as scalable architectures, APIs and secure access mechanisms. Using OpenAI Codex, teams executed end-to-end modernization of a credit card management platform by transforming COBOL business logic into a Java Spring Boot microservices architecture, developing a React-based user interface, migrating data models to PostgreSQL and implementing secure, JWT-based APIs. Codex supported the entire lifecycle, from code analysis and transformation to automated test generation and validation, ensuring functional parity with the legacy system. This AI-assisted approach delivered an estimated 80% productivity improvement compared to traditional methods, significantly reduced modernization timelines and risk and resulted in a more scalable, secure and future-ready system architecture.

- **Accelerating legacy codebase modernization:** A telco enterprise having a large legacy codebase, developed over two decades and supporting multiple generations of telecom technologies (2G to LTE/CUPS) through complex compilation flags and proprietary architectures, faced significant challenges due to the absence of original SMEs and outdated documentation. As a result, feature grooming became slow and heavily dependent on testing and debug logs to understand system behaviour. Codex was leveraged to automatically generate up-to-date high-level architecture views and module interaction insights directly from the latest code. These outputs were integrated into an internal E2E SDLC accelerator framework to enhance context engineering across development stages. This approach reduced code comprehension effort by 40–50%, accelerated the creation of software functional specifications, test plans and test cases by 30–40% and improved developer confidence in handling complex features. It also enabled knowledge democratization, improved design consistency and increased automation across SDLC workflows, leading to faster feature delivery and quicker resolution of production issues.

Risks, limitations and best practices

While the advantages of AI-assisted modernization are compelling, it is crucial to approach OpenAI Codex adoption with eyes open to the risks and limitations and follow best practice guidelines to mitigate these risks and ensure a successful, responsible implementation.

- **Security and vulnerabilities:** Studies have found that a notable portion of code suggestions from AI can have weaknesses. For instance, research found that around 25-30% of Copilot's generated code contained security vulnerabilities in one analysis. There's also the risk of secret leakage, where the AI might output sensitive data (e.g., credentials) if such patterns were in training data. It is advised to treat AI-generated code as potentially untrusted until reviewed by security teams. Integrating security scanners in the pipeline to catch common issues (SQL injection, XSS, etc.) and training developers to be aware of AI-specific pitfalls should be practiced and the final sign-off should be from humans.

- **Intellectual Property (IP) and licensing compliance:** Codex might occasionally produce code that is identical (or very similar) to open-source code it was trained on, potentially without proper attribution or incompatible licensing. It is recommended to use AI tools that have enterprise safeguards like filters to block suggestions that appear verbatim from public repositories or that have license text. Organizations should implement an IP review process until they are confident in compliance. Also, maintain an internal knowledge base of approved code patterns, possibly fine-tuning a model on your own code so that the AI is biased towards your proprietary implementations rather than anything from the wild.

- **Accuracy and hallucinations:** Codex, like any AI, can sometimes hallucinate, meaning that it might generate code that looks plausible but doesn't do what is needed or include subtle logic errors. Developers should not blindly accept Codex suggestions; instead encourage a culture of "trust but verify". In a modernization scenario, always compare the AI-generated module's output with the legacy module's output for various cases to ease the modernization journey.

- **Compliance and auditability:** In regulated industries (finance, healthcare, government), there may be concerns about using AI in software development. Questions about how decisions were made, or whether the AI's involvement needs to be documented for compliance reasons. Enterprises should maintain a clear audit trail of AI involvement. The easiest approach is to simply tag commits that had AI contribution or keep logs of Codex sessions. Prioritizing compliance early is a wise decision. Development should be able to explain and demonstrate how a piece of code was migrated with AI assistance and follow that up with comprehensive testing.

- **Model limitations and domain knowledge:** Codex is a general code model which means it may not know the specifics of a business' proprietary systems or business domain unless it is provided explicitly. Feed Codex with existing design documents or schematics as part of its input. This can be done via prompt or by embedding knowledge into a custom AGENTS.md as OpenAI suggests. It may also help to fine-tune your codebase to imbue it with domain knowledge. (Note: OpenAI allows fine-tuning for some models; or consider open-source alternatives). Always ensure a domain expert is the human in the loop to review AI output for domain correctness.

By following these best practices and proactively addressing the limitations, enterprises can significantly reduce the risks associated with adoption of Codex. In fact, many of these risk mitigations (like thorough reviews, better testing, audit trails) are just good software engineering practices that AI usage reinforces. When these steps are taken, the result is a modernized system that not only arrives faster, but is secure, compliant and robust.

Outlook

The rapid evolution of AI in software engineering suggests that what we see today with OpenAI Codex is just the beginning. While the AI-generated code may not be fully production-ready without review, it provides a substantial starting point, often covering 70-80% of the conversion automatically, thereby saving developers countless hours. Looking ahead, we can envision a future where AI-driven continuous modernization and even autonomous software engineering become part of the standard enterprise IT playbook.

In the future, modernization may shift from being a one-time project to an ongoing, continuous process aided by AI and organizations will heavily use AI tools to perform incremental upgrades constantly. Codex agents run in the background on code repositories, always monitoring outdated patterns, security issues, or performance anti-patterns and proactively suggesting improvements, preventing the build-up of massive technical debt. The future might see Codex-like agents owning entire user stories or defect fixes: for example, you could assign a bug to an AI agent that would then analyze the code, make the fix, run all tests and propose the change for merge, possibly with minimal human input beyond final approval. As confidence in these agents grows, the human role may shift more to oversight and intent specification while the AI handles the implementation. Developers might become more like product managers or architects for AI agents, describing high-level requirements and constraints, letting AI orchestrate the coding. However, even in this AI-forward future, human roles will remain vital, but at the same time they will evolve. For instance, leadership, using an AI provided analysis, decides what legacy components to modernize first. Leadership decides priorities. Ensuring that AI-driven changes align with business strategy will likely remain a human responsibility.

In conclusion, the trajectory for OpenAI Codex and similar AI technologies points toward an era of more autonomous, intelligent and continuous software modernization. Enterprises that today leverage Codex for accelerating migration are not only solving their immediate legacy problems but are also laying the groundwork for this AI-integrated future. By getting comfortable with AI in the development process now, establishing trust, governance and know-how, organizations will be well positioned to capitalize on further advances through modernization. The role of Codex in autonomous software engineering is likely to expand from an assistive role to a leading role for well-scoped tasks. However, the best outcomes will be achieved by those who treat AI as a powerful tool, one that amplifies human creativity and decision-making, rather than replaces it.

The role of HCLTech

HCLTech has partnered with OpenAI to deliver Codex-powered legacy modernization at scale, combining enterprise-grade AI with industrialized engineering services. This joint solution can help to accelerate transformation of aging application estates while reducing risk and total cost of ownership.

- **End-to-End modernization execution:** From portfolio assessment to production deployment, HCLTech manages the full modernization lifecycle: analysing legacy systems, designing modernization roadmaps, refactoring code and optimizing post-migration operations.
- **Modernization factory delivery model:** HCLTech's Codex-trained engineering squads would use AI-assisted workflows, templates, documentation, testing approaches etc. for refactoring and rationalization, achieving higher velocity and quality.
- **Modernization accelerators:** Dramatically shorten the modernization timelines by using a library of pre-built production-ready optimized prompts for common modernization tasks. These tasks include extracting business logic, rewriting COBOL to Java, generating API wrappers, refactoring monoliths into microservices etc. Any or all of these could dramatically shorten the modernization timelines.
- **Risk mitigation and governance:** HCLTech complements Codex's AI capabilities with robust delivery governance, ensuring security, compliance, coverage and architectural alignment in every modernization cycle.

Author

Saurabh Aggarwal,

PDEng, AI Evangelist and thought leader, HCLTech

Saurabh Aggarwal, PDEng is a GenAI and Agentic AI digital transformation leader at HCLTech, UK. With two decades of experience in providing advisory technology services to senior leaderships, he is helping businesses develop strategy to carve out the value preposition and leading their AI transformation journey to bring business improvements.

Thanks to Marcell Babai, Farinaz Shokrani, Dr. Pierre Erasmus, Kowsalya Subramanian, Dr. Aaron Johnson and Ashutosh Singh for their contribution.

HCLTech | Supercharging Progress™

hcltech.com