

# MÓDULO 5

Skills: diseño de SKILL.md y librería de habilidades

Reutilización · Gobernanza · Progressive Disclosure

*Plan de Formación — OpenAI Codex para Desarrolladores*

Nivel: Intermedio · Duración estimada: 3–4 horas

Versión 1.0 — Febrero 2026

## Índice

## 1. Introducción

Hasta ahora hemos aprendido a escribir prompts estructurados (Módulo 4) con constraints, Definition of Done y comandos de validación. Pero copiar y pegar esas plantillas en cada sesión es propenso a errores y no escala en equipos. Las Skills de Codex resuelven este problema: permiten encapsular conocimiento institucional en unidades reutilizables, compatibles y versionables.

Una skill es un pequeño paquete con un nombre, una descripción que indica cuándo usarla, y un cuerpo de instrucciones que Codex solo carga cuando la skill se activa. Este mecanismo de “progressive disclosure” mantiene el contexto ligero: Codex conoce qué skills existen, pero no carga sus instrucciones completas hasta que son necesarias.

En este módulo aprenderemos a diseñar, crear, organizar y probar skills que funcionen como estándar de equipo, sustituyendo los custom prompts (deprecados) por un sistema más potente y gobernable.

## 2. Objetivos de aprendizaje

#	Objetivo	Evidencia de logro
O1	Crear skills reutilizables y gobernables como estándar de equipo.	Produce 2 skills funcionales con YAML frontmatter válido.
O2	Dominar la activación explícita (\$nombre) e implícita (automática por descripción).	Demuestra ambos modos de invocación en el laboratorio.
O3	Diseñar descriptions precisas que eviten over-triggering.	La skill solo se activa en los prompts correctos.
O4	Comprender que los custom prompts están deprecados y migrar a skills.	Convierte un custom prompt a skill.

## 3. Contenidos teóricos

### 3.1 Qué es una skill

Una skill es un directorio con un fichero SKILL.md obligatorio y ficheros opcionales de soporte (scripts, referencias, templates). Codex inyecta en el contexto de ejecución solo el nombre, la descripción y la ruta de cada skill disponible. El cuerpo de instrucciones permanece en disco y no se inyecta a menos que la skill sea invocada.

#### 3.1.1 Estructura de un directorio de skill

```
mi-skill/
SKILL.md          # Obligatorio: instrucciones + metadatos
scripts/          # Opcional: código ejecutable
references/       # Opcional: documentación de apoyo
assets/           # Opcional: templates, recursos
agents/
    openai.yaml   # Opcional: interfaz y dependencias MCP
```

#### 3.1.2 Formato de SKILL.md

Cada SKILL.md tiene YAML frontmatter con dos campos obligatorios seguido de un cuerpo Markdown con instrucciones:

```
---
name: nombre-de-la-skill
description: Explica cuándo esta skill debe y no debe activarse.
---

## When to use this
Describe el escenario exacto.

## Instructions
Pasos imperativos que Codex debe seguir.

## Validation
Comandos de verificación que Codex ejecutará.
```

Campo	Obligatorio	Límite	Descripción
name	Sí	100 chars, 1 línea	Identificador único. Se invoca con \$name.
description	Sí	500 chars, 1 línea	Determina cuándo Codex activa la skill automáticamente. Clave para evitar over-triggering.
metadata	No	—	Campos adicionalesopcionales (short-description, etc.). Codex ignora claves extra.
Cuerpo (Markdown)	No	Sin límite fijo	Instrucciones detalladas. Solo se inyectan al invocar la skill.

 **Progressive Disclosure**

Codex solo carga el **nombre y description** de todas las skills en el contexto de ejecución. Las **instrucciones completas** (el cuerpo del SKILL.md) se inyectan únicamente cuando la skill es invocada. Esto mantiene el contexto ligero incluso con decenas de skills instaladas.

## 3.2 Ubicaciones y scopes

Las skills se cargan desde múltiples ubicaciones según quién debe acceder a ellas:

Scope	Ubicación	Viaja con el repo	Quién la usa
Repo (equipo)	.codex/skills/<nombre>/	Sí (versionada en Git)	Todos los que clonen el repo
User (personal)	~/.codex/skills/<nombre>/	No	Solo el usuario local, en todos sus repos
User (alternativo)	~/.agents/skills/<nombre>/	No	Igual que arriba (compatibilidad)
Admin/System	Definida por MDM o sistema	No	Máquinas gestionadas enterprise

**Recomendación:** Skills de equipo en .codex/skills/ (versionadas con Git). Skills personales en ~/.codex/skills/. Las skills del repo se aplican automáticamente cuando el proyecto está trusted.

### 💡 Requisito de trust

El proyecto debe estar marcado como **trusted** para que Codex cargue los ficheros .codex/ y AGENTS.md del repositorio. Si no está trusted, las skills de repo se ignoran silenciosamente.

## 3.3 Activación: explícita vs. implícita

### 3.3.1 Activación explícita (\$nombre)

El usuario escribe \$nombre-de-la-skill en el composer para invocarla directamente. Codex carga las instrucciones completas de la skill y las aplica a la tarea.

```
# Invocar skill explícitamente en el CLI o IDE:  
$ci-guardian  
  
# Invocar con contexto adicional:  
$ci-guardian  
El pipeline de CI está fallando en el paso de lint.  
Aquí está el log: @ci-output.log
```

### 3.3.2 Activación implícita (automática)

Si el prompt del usuario coincide con la description de una skill, Codex puede seleccionarla automáticamente sin que el usuario la invoque por nombre. La calidad de la description determina la precisión de este matching.

```
# Ejemplo: skill con description =  
# "Fix CI pipeline failures by analyzing build logs,  
# identifying the root cause, and applying a fix."  
  
# Este prompt ACTIVARÁ la skill automáticamente:  
"El CI está rojo. Arregla el error de lint del pipeline."  
  
# Este prompt NO DEBE activarla:  
"Añade una nueva feature de autenticación."
```

#### ⌚ Testing de triggering

Verifica que tu description es precisa probando con 3-4 prompts: 2 que **deben** activar la skill y 2 que **no deben** activarla. Si hay over-triggering, estrecha la description. Para validación sistemática la documentación recomienda evals con JSONL (codex exec --json).

## 3.4 Diseño de descriptions efectivas

La description es el campo más crítico de una skill. Determina cuándo Codex la activa automáticamente y cuándo no.

Calidad	Ejemplo de description	Problema / Beneficio
✗ Vaga	"Ayuda con el código del proyecto."	Over-triggering: se activa con cualquier prompt. Inútil.
✗ Amplia	"Mejora la calidad del código."	Se activa con refactors, reviews, tests, features... demasiado genérico.
✓ Precisa	"Fix CI pipeline failures by analyzing build logs, identifying root cause, and applying minimal fix."	Clara sobre QUÉ hace y CUÁNDO usarla.

✓ Con negación	"Draft commit messages in Conventional Commits format. Do not use for release notes or changelogs."	Explícita sobre cuándo NO activarse.
----------------	---	--------------------------------------

### 3.4.1 Reglas para buenas descriptions

- Describir la acción concreta: "Fix", "Draft", "Scaffold", "Review" (verbos imperativos).
- Incluir el contexto de activación: "when the user asks for help writing a commit message".
- Incluir negaciones si hay riesgo de confusión: "Do not use for...".
- Máximo 500 caracteres, una sola línea. Ser conciso pero específico.

## 3.5 Skills built-in y el ecosistema

Codex incluye skills integradas y un ecosistema público:

Skill	Descripción	Invocación
\$skill-creator	Crea nuevas skills interactivamente. Pregunta qué hace, cuándo activarse y tipo (instrucciones o script).	\$skill-creator
\$skill-installer	Instala skills desde el catálogo público de OpenAI.	\$skill-installer install <nombre>
\$create-plan (experimental)	Crea un ExecPlan (PLANS.md) para tareas complejas. Necesita instalación.	\$skill-installer install create-plan
\$plan	Genera un plan de implementación con milestones.	\$plan <descripción>

- Catálogo público: [github.com/openai/skills](https://github.com/openai/skills)
- Comunidad y especificación: [agentskills.io](https://agentskills.io)

## 3.6 Custom prompts (deprecados) vs. skills

Los custom prompts son un mecanismo legacy que vive en `~/.codex/` y requiere invocación explícita vía `/prompts:nombre`. Están marcados como deprecados por OpenAI. Las skills los sustituyen completamente:

Característica	Custom Prompts (deprecados)	Skills
Ubicación	Solo <code>~/.codex/</code> (personal)	<code>~/.codex/skills/</code> (personal) + <code>.codex/skills/</code> (repo)
Compartir con equipo	No (no se versionan)	Sí (versionadas en Git con el repo)
Activación implícita	No (solo explícita vía <code>/prompts:*</code> )	Sí (automática según description)
Ficheros de soporte	No	Sí ( <code>scripts/</code> , <code>references/</code> , <code>assets/</code> )
MCP dependencies	No	Sí (vía <code>agents/openai.yaml</code> )
Progressive disclosure	No (siempre se cargan)	Sí (solo nombre+description en contexto)

### ⚠️ Migración obligatoria

Si tu equipo usa custom prompts (ficheros .md en `~/.codex/`), **migra a skills**. El proceso es simple: mueve el contenido a un SKILL.md con YAML frontmatter (name + description) en la ubicación correspondiente.



## 3.7 Configuración avanzada

### 3.7.1 Deshabilitar una skill sin borrarla

```
# En ~/.codex/config.toml:
[[skills.config]]
path = "/ruta/a/la/skill/SKILL.md"
enabled = false

# Reiniciar Codex después de cambiar config.toml
```

### 3.7.2 Configurar interfaz y dependencias MCP (agents/openai.yaml)

```
# mi-skill/agents/openai.yaml
interface:
  display_name: "CI Guardian"
  short_description: "Fix CI failures automatically"
  icon_small: "./assets/ci-icon.svg"
  brand_color: "#E53E3E"
  default_prompt: "Analiza el log de CI y corrige el error."

dependencies:
  tools:
    - type: "mcp"
      value: "github"
      description: "GitHub MCP para acceder a PRs y CI"
```

### 3.7.3 Detección de cambios en caliente

Codex detecta cambios en ficheros de skills sin necesidad de reiniciar (live skill update detection). Al editar un SKILL.md, los cambios se aplican en la siguiente invocación.

## 4. Buenas prácticas

### 4.1 Skills por disciplina

Organizar las skills del equipo por área de responsabilidad:

Disciplina	Skill sugerida	Propósito
Backend API	\$api-endpoint	Scaffoldar endpoint REST con validación, tests y docs
Frontend React	\$react-component	Crear componente con tipos, storybook y tests
Base de datos	\$db-migration	Crear migración con rollback y verificación de schema
Seguridad	\$security-review	Auditar código buscando vulnerabilidades comunes
CI/CD	\$ci-guardian	Diagnosticar y corregir fallos de pipeline
Documentación	\$update-docs	Actualizar documentación tras cambios de código
Code Review	\$review-checklist	Revisar PR con checklist de calidad estándar

### 4.2 Incluir checklist de verificación en cada skill

Toda skill debería terminar con una sección de validación:

```
## Validation Checklist
Before finishing, verify all of the following:
1. Run tests: `pytest tests/ -v`
2. Run linter: `flake8 src/`
3. Check formatting: `black --check src/`
4. Verify diff scope: `git diff --stat` (only expected files)
5. Show all results.
```

### 4.3 Mantener skills pequeñas y composable

- Una skill = un trabajo. No mezclar “corregir CI” con “revisar seguridad”.
- Si una skill crece demasiado, dividirla en skills más específicas.
- Preferir instrucciones sobre scripts. Usar scripts solo cuando se necesita comportamiento determinista o herramientas externas.
- Escribir instrucciones imperativas con inputs y outputs explícitos.
- Asumir que Codex no tiene contexto previo: escribir instrucciones autocontenido.

### 4.4 Versionar skills como código

- Skills de equipo en .codex/skills/ versionadas con Git.
- Code review obligatorio para cambios en skills (como cualquier otro código).
- Incluir tests de triggering en el proceso de revisión.

## 5. Errores comunes y cómo evitarlos

### 5.1 Descriptions vagas (“ayuda con todo”)

**El problema:** Una description como “Help with coding tasks” se activa con cualquier prompt. Contamina el contexto con instrucciones irrelevantes.

**La solución:** Description precisa con verbo de acción, contexto de activación y negaciones: “Fix CI pipeline failures by analyzing build logs and applying minimal fixes. Do not use for feature development or code review.”

### 5.2 Mezclar políticas de seguridad con estilo

**El problema:** Una sola skill que combina “usa camelCase” con “nunca hardcodees secrets”. Las prioridades no están claras.

**La solución:** Separar: convenciones de estilo en AGENTS.md (contexto permanente); auditorías de seguridad como skill de invocación explícita (\$security-review).

### 5.3 Skills gigantes no componibles

**El problema:** Una skill de 200 líneas que cubre deployment + testing + monitoring. Al invocarla se inyecta todo aunque solo se necesite una parte.

**La solución:** Dividir en skills enfocadas: \$deploy, \$run-tests, \$setup-monitoring. Cada una hace un solo trabajo bien.

### 5.4 Tabla de errores adicionales

Error	Consecuencia	Prevención
name o description multi-línea	Codex ignora la skill con error de validación	name: 1 línea, ≤100 chars. description: 1 línea, ≤500 chars.
Fichero no se llama SKILL.md	Codex no detecta la skill	Mayúscula exacta: SKILL.md
Directorio de skill es symlink	Codex ignora symlinks	Usar directorios reales, no symlinks
Seguir usando custom prompts	No se comparten con equipo, no se activan automáticamente	Migrar a skills (custom prompts están deprecados)
No reiniciar tras añadir skill	La skill no aparece (en versiones anteriores)	La detección en caliente es reciente; ante la duda, reiniciar

## 6. Casos de uso reales

### 6.1 Skill “fix CI hasta verde”

**Escenario:** Pipeline falla frecuentemente por lint, tests rotos o dependencias. El equipo quiere una skill que diagnostique y corrija automáticamente con límites claros.

#### SKILL.md implementado

```
---
name: ci-guardian
description: Fix CI pipeline failures by analyzing build logs, identifying root cause, and applying a minimal fix. Use when CI is red. Do not use for new features or code review.
---

## When to use this
Use when a CI pipeline has failed and the user wants to diagnose and fix it.

## Workflow
1. Ask the user for the CI log or read @ci-output.log if available.
2. Identify the failing step (lint, test, build, deploy).
3. Identify the root cause in the codebase.
4. Apply a minimal fix (smallest possible change).
5. Do NOT change unrelated code.

## Constraints
- Maximum 3 files modified.
- No new dependencies.
- No changes to public APIs.
- If the fix requires more than 30 lines, STOP and ask the user.

## Validation
1. Run the same CI command that failed.
2. Run the full test suite.
3. Show `git diff --stat`.
4. Confirm CI would pass with these changes.
```

**Resultado:** Los desarrolladores escriben “el CI está rojo” y Codex activa la skill automáticamente. El fix es mínimo, acotado y verificado.

### 6.2 Skill “migración legacy” con estrategia incremental

**Escenario:** Equipo migrando de Flask a FastAPI. Necesitan una skill que migre endpoints uno a uno, preservando la funcionalidad.

#### SKILL.md implementado

```
---
name: migrate-flask-to-fastapi
description: Migrate a Flask endpoint to FastAPI preserving behavior. Use when the user specifies which endpoint to migrate. Do not use for bulk migrations or non-Flask code.
---
```

```
## When to use this
Use when migrating a single Flask endpoint to FastAPI.
The user should specify which endpoint to migrate.

## Strategy: Incremental Migration
1. Read the Flask endpoint code (@old_endpoint).
2. Identify: route, HTTP method, request parsing, response format.
3. Create equivalent FastAPI endpoint with Pydantic models.
4. Preserve exact same behavior (same status codes, same response).
5. Create/update tests to verify equivalence.

## Constraints
- One endpoint per invocation (atomic migration).
- Do not modify other endpoints.
- Do not remove the Flask endpoint (coexistence until verified).
- Add deprecation comment to the Flask version.

## Validation
1. Both Flask and FastAPI endpoints respond identically.
2. Tests pass for both versions.
3. Show `git diff --stat` (should be small and scoped).
```

## 7. Laboratorio práctico (L5) — Crear 2 skills corporativas

### ⌚ Objetivo del laboratorio

**Crear 2 skills funcionales y listas para producción:** una de CI (\$ci-guardian) y una de refactorización segura (\$refactor-safe). Probar invocación explícita (\$nombre) y verificar que la description es precisa.

### 7.1 Prerrequisitos

- Codex CLI instalado y autenticado (Módulos 1-2).
- Python 3.10+ con venv.
- Git instalado.

### 7.2 Paso 1 — Crear proyecto base

*Tiempo estimado: 5 minutos*

```
mkdir /tmp/codex-lab05 && cd /tmp/codex-lab05
git init

# Crear entorno virtual
python3 -m venv .venv
source .venv/bin/activate      # Windows: .venv\Scripts\activate
pip install pytest flake8

cat > .gitignore << 'EOF'
.venv/
__pycache__/
*.pyc
EOF

mkdir -p src tests

cat > src/calculator.py << 'PYEOF'
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    return a / b  # Bug: no maneja división por cero
PYEOF

cat > tests/test_calculator.py << 'PYEOF'
from src.calculator import add, subtract, multiply, divide

def test_add():
    assert add(1, 2) == 3
    assert add(-1, 1) == 0
    assert add(0, 0) == 0
```

```

    assert add(2, 3) == 5

def test_subtract():
    assert subtract(5, 3) == 2

def test_divide():
    assert divide(10, 2) == 5.0
PYEOF

# Verificar que los tests pasan
PYTHONPATH=. pytest tests/ -v

git add -A && git commit -m "chore: scaffold lab05"

```

## 7.3 Paso 2 — Crear skill \$ci-guardian

*Tiempo estimado: 10 minutos*

```

# Crear directorio de skill en el repo
mkdir -p .codex/skills/ci-guardian

cat > .codex/skills/ci-guardian/SKILL.md << 'EOF'
---
name: ci-guardian
description: Fix CI pipeline failures by analyzing logs, identifying root cause,
and applying minimal fix. Do not use for feature development or code review.
---

## When to use this
Use when CI has failed and the user wants to diagnose and fix it.
The user will provide CI logs or error output.

## Workflow
1. Read the CI log or error output provided by the user.
2. Identify the failing step (lint, test, build).
3. Trace the root cause to the specific file and line.
4. Apply a minimal fix (smallest possible change).
5. Run the failing command to verify the fix.

## Constraints
- Maximum 3 files modified.
- No new dependencies.
- No changes to public function signatures.
- If the fix requires more than 30 lines of new code, STOP
  and explain the situation to the user instead.

## Validation Checklist
1. Run the exact CI command that originally failed.
2. Run the full test suite: PYTHONPATH=. pytest tests/ -v
3. Run linter: flake8 src/
4. Show git diff --stat to confirm minimal scope.
EOF

```

```
git add .codex/ && git commit -m "skill: añadir ci-guardian"
```

## 7.4 Paso 3 — Crear skill \$refactor-safe

*Tiempo estimado: 10 minutos*

```
mkdir -p .codex/skills/refactor-safe

cat > .codex/skills/refactor-safe/SKILL.md << 'EOF'
---
name: refactor-safe
description: Refactor code safely with no behaviour change. Runs tests before and
after to guarantee equivalence. Do not use for adding features or fixing bugs.
---

## When to use this
Use when the user wants to refactor existing code for clarity,
reduce complexity, or extract functions WITHOUT changing the
external behavior.

## Invariant
The external behavior must NOT change. All existing tests must
pass without modification. Public function signatures must not
be altered.

## Workflow
1. Run all tests FIRST to establish the green baseline.
2. Read the target code and identify refactoring opportunities.
3. Propose a plan (what to extract/rename/simplify).
4. Implement the refactoring.
5. Add type hints to new functions.
6. Run all tests again to verify NO BEHAVIOUR CHANGE.

## Constraints
- Only modify files explicitly mentioned by the user.
- Do not change public function signatures or return types.
- Do not add new dependencies.
- Maximum 50 lines of new code.
- Add tests for extracted functions.

## Validation Checklist
1. Run tests before: PYTHONPATH=. pytest tests/ -v (MUST pass)
2. Apply refactoring.
3. Run tests after: PYTHONPATH=. pytest tests/ -v (MUST pass)
4. Run linter: flake8 src/
5. Show git diff --stat (scope check).
6. Confirm: all original tests pass WITHOUT modification.
EOF

git add .codex/ && git commit -m "skill: añadir refactor-safe"
```

## 7.5 Paso 4 — Probar invocación explícita

*Tiempo estimado: 10 minutos*

Asegúrate de que el venv está activo y lanza Codex:

```
cd /tmp/codex-lab05
source .venv/bin/activate
codex --full-auto
```

### TEST 1: Invocación de \$ci-guardian. Escribe en el TUI:

```
$ci-guardian

Añadí este test que falla:
def test_divide_by_zero():
    assert divide(10, 0) == "error"

El CI está rojo por este test. Arregla el código para que pase.
```

Observar que Codex carga las instrucciones de ci-guardian y aplica un fix mínimo (probablemente añadir manejo de ZeroDivisionError en divide()).

### TEST 2: Invocación de \$refactor-safe (tras commit del fix anterior). Prompt:

```
$refactor-safe

Refactoriza @src/calculator.py:
- Extrae la validación de inputs a una función auxiliar.
- Añade type hints a todas las funciones.
```

Observar que Codex ejecuta tests ANTES y DESPUÉS del refactor (como indica la skill) y verifica que el diff es acotado.

## 7.6 Paso 5 — Verificar resultado

Verificación	Resultado esperado	Comando
Skills detectadas por Codex	ci-guardian y refactor-safe listadas	\$ en el composer muestra las skills
\$ci-guardian fixó el bug	divide() maneja ZeroDivisionError	PYTHONPATH=. pytest tests/ -v
\$refactor-safe no rompió nada	Todos los tests pasan sin modificar tests	PYTHONPATH=. pytest tests/ -v
Diff acotado	Solo src/calculator.py y tests/	git diff --stat
SKILL.md válido	Sin errores YAML al arrancar Codex	Codex arranca sin warnings

## 7.7 Limpieza (OBLIGATORIA)

### ⚠️ Limpieza de recursos

Ejecutar al finalizar el laboratorio:

```
# 1. Desactivar entorno virtual
deactivate
```

```
# 2. Borrar el repositorio
rm -rf /tmp/codex-lab05

# 3. Verificar
ls /tmp/codex-lab05 2>/dev/null && echo 'ERROR' || echo 'OK: limpio'

# 4. Si creaste skills personales de prueba en ~/.codex/skills/,
#     borrar las carpetas correspondientes.
```

## 8. Resumen del módulo y siguientes pasos

### 8.1 Conceptos clave aprendidos

Concepto	Detalle
Skill	Directorio con SKILL.md (name + description + instrucciones) + ficheros opcionales
Progressive disclosure	Solo name/description en contexto; instrucciones solo al invocar
Activación explícita	\$nombre en el composer
Activación implícita	Automática según match con description
Ubicaciones	.codex/skills/ (repo)   ~/.codex/skills/ (personal)
Custom prompts	Deprecados. Migrar a skills.
\$skill-creator	Skill built-in para crear nuevas skills interactivamente
Description precisa	Verbo de acción + contexto + negaciones para evitar over-triggering
agents/openai.yaml	Configuración de interfaz y dependencias MCP

### 8.2 Preparación para el Módulo 6

En el Módulo 6 abordaremos MCP (Model Context Protocol): cómo conectar Codex a herramientas externas (bases de datos, APIs, servicios) para ampliar sus capacidades. Para prepararse:

- Revisar la documentación de MCP: <https://developers.openai.com/codex/mcp/>
- Pensar en qué herramientas externas usa tu equipo que podrían beneficiarse de integración con Codex.
- Explorar el catálogo de skills: [github.com/openai/skills](https://github.com/openai/skills)

## 9. Referencias y recursos

- Skills overview: <https://developers.openai.com/codex/skills/>
- Create skills: <https://developers.openai.com/codex/skills/create-skill/>
- Custom Prompts (deprecated): <https://developers.openai.com/codex/custom-prompts/>
- Testing Agent Skills with Evals: <https://developers.openai.com/blog/eval-skills/>
- Agent Skills Specification: [agentskills.io](https://agentskills.io)
- Skills catalog: [github.com/openai/skills](https://github.com/openai/skills)
- Codex Workflows: <https://developers.openai.com/codex/workflows/>
- AGENTS.md guide: <https://developers.openai.com/codex/guides/agents-md/>
- Automations (Codex App): <https://developers.openai.com/codex/app/automations/>
- Codex changelog: <https://developers.openai.com/codex/changelog/>