0x06tudio

Progetto didattico di Ingegneria del Software
Università degli Studi di Padova



0x06tudio@gmail.com

Sistemi di Harness e Gestione del Contesto AI

Studio Tecnologico – Capitolato C9: SpecCheck (Zucchetti S.p.A.)

Informazioni sul documento

Versione:	0.1.0
Stato:	Approvato
Data ultimo aggiornamento:	2026/11/18
Uso:	Interno
Destinatari:	0x06tudio

Registro delle modifiche

Versione	Data	Descrizione	Autore	Verificatore
0.1.0	2026/11/18	Prima stesura del documento	Davide Colabove	Nicolò Beltrame

Indice

1 Introduzione **3**
1.1 Scopo del documento . . . 3
1.2 Riferimenti . . . 3
1.2.1 Riferimenti normativi . . . 3
1.2.2 Riferimenti informativi . . . 3

2 Contesto e Sfide dell'Infrastruttura AI **4**

3 Analisi delle Tecnologie Chiave **4**
3.1 Architettura dei Sistemi di Harness . . . 4
3.2 Gestione del Contesto a Lungo Termine . . . 4
3.3 Architettura delle Skills e Sidecar Executables . . . 4

4 Architettura Proposta e Proof of Concept (PoC) **5**

5 Valutazione Comparativa delle Metodologie **6**

6 Conclusioni e Raccomandazioni per il Progetto **7**

1 Introduzione

1.1 Scopo del documento

Il presente studio tecnologico analizza l'architettura dei sistemi di $Harness_G$ e le metodologie di *Gestione del Contesto* AI_G, con un focus specifico sull'estensione dell'agente a riga di comando **OpenCode**. L'obiettivo è definire l'infrastruttura tecnologica locale, i pattern di integrazione e l'architettura delle *Skill* necessarie per implementare il modulo di tracciabilità end-to-end richiesto dal progetto *SpecCheck*.

Il documento stabilisce le basi per superare i limiti dell'approccio generativo non strutturato, fornendo all'agente gli strumenti per governare progetti software complessi attraverso vincoli architetturali persistenti e tool di validazione deterministica.

1.2 Riferimenti

1.2.1 Riferimenti normativi

- Capitolato d'appalto C9: *SpecCheck – Sviluppo di Skills per OpenCode per la verifica della tracciabilità end-to-end da casi d'uso a requisiti e codice sorgente* (Zucchetti S.p.A., A.A. 2026/2027);
- *Norme di Progetto* del gruppo *0x06tudio*.

1.2.2 Riferimenti informativi

- **Agente OpenCode:** `https://opencode.ai/`;
- **Ecosistema Harness:** *Claude Code* (`https://claude.com/it/product/claude-code`), *Deepseek harness*;
- **Benchmark:** *SWE-Bench* (`https://github.com/swe-bench/SWE-bench`) e *Terminal-bench* (`https://www.tbench.ai/`);
- **Agent Skills & Framework:** *Agent-skills* (Addy Osmani), *SuperPowers*, *OpenSpec*.

2 Contesto e Sfide dell'Infrastruttura AI

Il capitolato C9 evidenzia una transizione fondamentale nell'ingegneria del software assistita dall'Intelligenza Artificiale. Questa transizione pone tre criticità architetturali primarie per il team di sviluppo:

1. **Fallimento del "Vibe Coding":** L'approccio puramente generativo (trasmettere idee e sensazioni a un LLM via chatbot) si è dimostrato inadeguato. Produce infatti codice affetto da errori, disconnesso dai requisiti formali, a causa dell'incapacità di mantenere la struttura del flusso di informazioni nel tempo.

2. **Ritenzione del Contesto Architetturale:** A differenza della singola interazione generativa, le scelte architetturali di un progetto reale devono essere documentate e vincolate a lungo termine. I sistemi di Harness moderni incidono per circa il 30% sulle performance di benchmark complessi (es. SWE-bench) proprio grazie alla loro capacità di mantenere questa impalcatura di regole intatta.

3. **Esecuzione Locale e Vincoli di Modello:** L'azienda committente richiede la compatibilità del sistema di tracciabilità anche con modelli ridotti (SLM) come **Gemma 4 26B** e **Qwen 3.8 27B**, utilizzabili senza limitazioni di costo. Ciò impone la progettazione di un'infrastruttura capace di interfacciarsi con motori di inferenza locali, riducendo il carico cognitivo dell'LLM tramite deleghe di precisione a eseguibili esterni.

3 Analisi delle Tecnologie Chiave

3.1 Architettura dei Sistemi di Harness

Un sistema di Harness funge da livello di orchestrazione tra il *Large Language Model (LLM)*$_G$, il file system del progetto e gli strumenti del sistema operativo. OpenCode opera nativamente tramite un'interfaccia a riga di comando (CLI/TUI) che pianifica ed esegue attività in autonomia. Per ottimizzare le prestazioni con i modelli locali richiesti dal capitolato (es. Qwen 3.8), l'infrastruttura del nostro progetto dovrà sfruttare motori di inferenza open-source (come `llama-server` o `oMLX`) configurati per la quantizzazione avanzata della KV cache. Questo approccio architetturale garantisce un'allocazione della memoria rigorosa e un throughput elevato.

3.2 Gestione del Contesto a Lungo Termine

La memoria di un progetto software non può risiedere esclusivamente nella *context window* dell'LLM. L'approccio consolidato di OpenCode prevede l'uso di file strutturali persistenti:

- ***AGENTS.md***$_G$**:** Un file guida che istruisce il sistema sulle convenzioni architetturali, sulle librerie in uso e sui vincoli del dominio, mantenendo integre le scelte durante tutto il ciclo di sviluppo.

- **System Prompt Dinamici:** Il sistema di harness è in grado di iniettare porzioni specifiche di contesto (ad esempio, l'esito di una validazione di tracciabilità) direttamente nel prompt di sistema dell'agente prima di una nuova operazione di sviluppo.

3.3 Architettura delle Skills e Sidecar Executables

L'estensibilità è il cuore pulsante di OpenCode. Il sistema viene potenziato tramite la tecnica delle *Skills*$_G$: file di risorse che definiscono in linguaggio naturale e formale i passaggi necessari

per completare un task non supportato nativamente (come la validazione dei requisiti del nostro capitolato).

Poiché attività come il parsing preciso di diagrammi o il rilevamento di discrepanze logiche tra specifiche e codice non possono essere affidate al determinismo fluttuante di un LLM, le skill devono necessariamente interfacciarsi con dei ***Sidecar Executables***$_G$. Si tratta di binari CLI_G esterni, sviluppati ad hoc (ideali linguaggi compilati a zero dipendenze e alte performance come Go o Rust), che vengono richiamati dall'agente per eseguire algoritmi di *Semantic Matching* rigidi.

4 Architettura Proposta e Proof of Concept (PoC)

Per implementare la pipeline di tracciabilità richiesta dal capitolato, si propone un'architettura disaccoppiata basata su **YAML/JSON per le definizioni delle Skill** e **Python/Go per gli eseguibili Sidecar**. Questo stack garantisce esecuzioni a terminale estremamente rapide, un requisito essenziale per mantenere la reattività della TUI_G di OpenCode.

```yaml
name: "speccheck-validation-skill"
description: "Avvia il Sidecar CLI per verificare la tracciabilita tra i requisiti formali e il codice sorgente."
trigger: "post_code_generation"
required_context:
  - "docs/requisiti.md"
  - "src/**/*.go"
execution:
  type: "local_command"
  command: "./speccheck-cli"
  args: ["verify", "--target", "src", "--specs", "docs/requisiti.md", "--output", "json"]
on_result:
  success: "Update AGENTS.md status and output OK to TUI"
  failure: "Render TUI warning via OpenCode API and log Discrepancies"
```

Listing 1: Definizione concettuale di una Skill OpenCode per la validazione della tracciabilità (`speccheck-validation-skill.yaml`).

```go
package main

import (
    "encoding/json"
    "fmt"
    "os"
)

// Result definisce la struttura dati strutturata restituita all'Harness OpenCode
type Result struct {
    Compliant     bool     `json:"compliant"`
    Discrepancies []string `json:"discrepancies"`
}

func main() {
    // 1. Parsing degli argomenti (flag CLI) iniettati automaticamente da OpenCode
    // 2. Lettura deterministica dell'AST del codice sorgente analizzato
```

```
    // 3. (Opzionale) Chiamata API locale al modello Qwen/Gemma per inferenza semantica incrociata

    // Mockup del risultato della validazione
    result := Result{
        Compliant: false,
        Discrepancies: []string{"Fallimento tracciabilita: UC-03 risulta assente nell'implementazione di auth.go"},
    }

    // Stampa in stdout formattata (JSON) per essere parsata e consumata dall'Agente OpenCode
    jsonOut, _ := json.Marshal(result)
    fmt.Println(string(jsonOut))

    // Segnala formalmente il fallimento dell'azione alla Skill
    os.Exit(1)
}
```

Listing 2: Struttura base del Sidecar Executable (CLI) scritto in Go e invocato dalla Skill OpenCode.

5 Valutazione Comparativa delle Metodologie

Criterio	**Vibe Coding (Chatbot Puro)**	**Scripting Tradizionale (CI/CD)**	**Harness + Skills (Approccio Spec-Check)**
Ritenzione del Contesto	Volatile (fortemente limitata alla context window corrente).	Nulla (limitata alla pura esecuzione di unit test e linter).	**Permanente** e garantita da file strutturali come `AGENTS.md`.
Determinismo	Basso (elevato rischio di allucinazioni architetturali e invenzioni).	Assoluto (ma incapace di eseguire analisi semantica sui requisiti in linguaggio naturale).	**Ibrido** (Sfrutta l'astrazione semantica dell'LLM combinata alla logica di un Sidecar).
Estensibilità	Vincolata unicamente ai plugin forniti dal vendor o dalla piattaforma.	Alta (ma richiede manutenzione manuale costante delle pipeline).	**Totale** (integrazione fluida e dinamica di Tool esterni e Skill native).
Costi di Inferenza	Alti (necessità di continui re-prompting per aggiustare il tiro).	Nulli.	**Ottimizzati** (grazie a chiamate LLM mirate e all'uso di SLM locali gratuiti).

6 Conclusioni e Raccomandazioni per il Progetto

Lo strumento OpenCode, operando come orchestratore di contesto, rappresenta l'impalcatura ideale per risolvere il problema della tracciabilità end-to-end posto dal capitolato. Per garantire l'efficacia, la modularità e la facilità di installazione del prodotto finale richiesto da Zucchetti, si delineano le seguenti raccomandazioni per le fasi di sviluppo:

1. **Fase PoC (Proof of Concept):** Concentrare gli sforzi iniziali sull'ingegnerizzazione dei file di configurazione delle *Skills* (in YAML) e sulla solidificazione della comunicazione bidirezionale (I/O in formato JSON) tra OpenCode e i primi script Sidecar, delegati alla lettura e al parsing dei diagrammi UML testuali.
2. **Fase MVP (Minimum Viable Product):** Consolidare i Sidecar CLI (utilizzando preferibilmente linguaggi compilati per massimizzare le prestazioni) e sviluppare lo script unificato di automazione (`bash`/`powershell`). Questo script si occuperà dell'installazione automatizzata e trasparente dell'intero pacchetto $toolchain_G$ *SpecCheck* nell'ambiente OpenCode dell'utente finale.
3. **Integrazione Dashboard (Requisito Opzionale):** Poiché la CLI/TUI nativa di OpenCode presenta limitazioni fisiologiche per l'esplorazione complessa delle discrepanze rilevate, il Sidecar dovrà essere progettato per generare file di stato strutturati. Questi file saranno successivamente intepretati da un'interfaccia HTML/Web esterna, soddisfacendo così in modo elegante il requisito opzionale di dashboarding richiesto dal committente.