T.C. SANAYİ VE TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK BİLGEM

# KODLAMA AJANI KULLANICILARI İÇİN HARNESS MÜHENDİSLİĞİ

ARAŞTIRMA SERİSİ - SAYI 42



**TÜBİTAK BİLGEM**

YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

# KODLAMA AJANI KULLANICILARI İÇİN HARNESS MÜHENDİSLİĞİ

**ARAŞTIRMA SERİSİ - SAYI 42**


**Yazar**

Muhsin ÖZBEK

**Yayın Koordinatörü**

Nehir BAYAR KİBAR

**Editörler**

Özay DUMAN

Muhammed Fatih DOĞMUŞ

Tuğçe YILMAZ

Sinem ALTUĞ GÖL

**Tasarım**

Ahmet YİĞİT



©2026 - Tüm hakları saklıdır.



İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

https://bilgem.tubitak.gov.tr/yte/



Yayımlanan yazıların sorumluluğu yazarlarına aittir, TÜBİTAK BİLGEM sorumlu tutulamaz.

# Ön Söz




*TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE), 2012 yılından bu yana yazılım teknolojilerinde Ar-Ge faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.*

*TÜBİTAK BİLGEM YTE tarafından hazırlanan Araştırma Serisi ile kurum içi içerik üretme çalışmalarının yaygınlaştırılması ve hazırlanan içeriklerin sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanılması hedeflenmektedir.*

# İçindekiler


6 **Giriş**

**Harness Mühendisliğinin Kavramsal Temelleri ve Alandaki Çalışmalar**

9 **Harness Kavramı ve Tarihsel Gelişimi**

Model ve Ajan Arasındaki Fark
Harness'ın Doğuşu ve Dönüşümü

10 İstem Mühendisliğinden Harness Mühendisliğine

**Harness'ın Anatomisi**

11 Ajan Döngüsü
Sistem İstemi

12 Araç Katmanı
Çalışma Ortamı ve Yürütücü
Uzak ve Bulut Tabanlı Ajanlar

14 Katmanların Bütüncül Görünümü: ETCLOVG Taksonomisi

15 **Bağlam Mühendisliği**

Tam Zamanında Bağlam ve Ajansal Arama
Sıkıştırma (Compaction) ve Yapılandırılmış Not Alma (Structured Note-Taking)

16 Kalıcı Bağlam: AGENTS.md ve Hafıza

**Araç Tasarımı ve Protokoller**

Model Context Protocol (MCP)

17 Araçların Harness İçindeki Rolü

18 MCP Araçları için Tasarım İlkeleri

19 Language Server Protocol (LSP)
Agent Client Protocol (ACP)

20 **Geri Bildirim Döngüleri ve Doğrulama**

Test, Lint ve Mimari Kural Tabanlı Doğrulama

21 Plan Modu ve İnsan Onayı Kapıları
Doğrulayıcı Ajanlar ve Eleştirmen Desenleri

**Güvenlik ve İzin Modeli**

İzin Modelleri ve Otomasyon Seviyeleri

22 Sanal Alan (Sandbox) Mimarileri
İstem Enjeksiyonu ve Ölümcül Üçlü

23 **Çoklu Ajan Mimarileri**

Alt Ajanlar (Subagents)

24 Orkestrasyon Desenleri

25 Ne Zaman Çoklu Ajan?

26 **Değerlendirme ve Gözlemlenebilirlik**
Değerlendirme (Evals)

27 İzleme, Telemetri ve Maliyet

28 **Kullanıcılar için Harness Yapılandırma Pratikleri**
AGENTS.md ve Proje Bağlamının Kurulması

29 Özelleştirilmiş Komutlar ve Yetenekler (Skills)

30 İzin ve Otomasyon Seviyesinin Ayarlanması

31 Ekip Düzeyinde Harness Standartlaşması
**Örnek Ajan Uygulaması İncelemesi: OpenCode**
Genel Bakış

32 Harness Anatomisi Açısından OpenCode Yapılandırma Pratikleri

33 Diğer Ajanlarla Karşılaştırma

34 **Sonuç ve Öneriler**

35 **Kaynakça**

37 **Kısaltmalar**

38 **Şekiller ve Tablolar**

# Giriş

Kodlama ajanları son iki yılda yazılım geliştirme pratiğinin merkezine yerleşmiştir. Fakat bu araçların gücünün nereden kaynaklandığı çoğu zaman yeterince açıklanmamaktadır. Aynı LLM [Large Language Model (Büyük Dil Modeli)]'ni kullanan iki ajan, aynı görevde bambaşka sonuçlar üretebilmektedir. Bu farkın kaynağı modelin kendisi değil; modelin etrafını saran, ona araçlarını, bağlamını, geri bildirim döngülerini ve güvenlik sınırlarını sağlayan yapı, yani "harness" (koşum) katmanıdır. Harness mühendisliği, ajanın ham model yeteneğini güvenilir mühendislik çıktısına dönüştüren disiplinin adıdır [18]. Anthropic, OpenAI, Microsoft ve Google'ın 2024–2026 arasında yayımladığı mühendislik raporları, ajan performansındaki iyileşmelerin giderek artan bir bölümünün model güncellemelerinden değil, harness iyileştirmelerinden geldiğini göstermektedir ([2], [9], [15], [17]).

Bu çalışmada harness kavramı, model ile ajan arasındaki fark ve harness'ın anatomisi ele alınmaktadır. Bağlam mühendisliği ve hafıza yönetiminin yanı sıra araç tasarımı ilkeleri ile MCP [Model Context Protocol (Model Bağlam Protokolü)], LSP [Language Server Protocol (Dil Sunucusu Protokolü)] ve ACP [Agent Client Protocol (Ajan İstemci Protokolü)] standartları incelenmektedir. Ayrıca geri bildirim döngüleri ve doğrulama mekanizmaları, izin ve güvenlik modelleri, çoklu ajan mimarileri, değerlendirme (evals) ve gözlemlenebilirlik pratikleri ile kodlama ajanı kullanıcılarının harness'larını kendi projelerine göre nasıl yapılandırabilecekleri ele alınmaktadır.

Birincil kaynak taramasına dayanan nitel bir derleme niteliği taşıyan çalışma kapsamında Anthropic, OpenAI, Microsoft ve Google gibi kuruluşların 2024–2026 döneminde yayımladığı mühendislik yayınları, protokol belirtimleri ve açık kaynaklı ajan belgeleri incelenmiş ([2], [6], [7], [15], [17], [22]); elde edilen bulgular açık kaynaklı OpenCode ajanı üzerinde somutlaştırılmıştır. Kodlama ajanlarıyla sınırlı tutulan incelemede, harness bileşenlerinin karşılaştırmalı başarım ölçümü ve deneysel doğrulaması kapsam dışında bırakılmıştır.

Bu çalışmanın amacı, kullanıcıların harness katmanını çözümleyerek kendi projelerine göre yapılandırabilmeleri için gerekli kavramsal çerçevenin ve pratik ilkelerin bütüncül biçimde sunulmasıdır. Böylece harness bileşenlerinin çıktı kalitesine etkisinin açıklanması ve daha güvenilir, öngörülebilir ve yüksek kaliteli yazılım geliştirme süreçlerine katkı sağlanması amaçlanmaktadır.

# Harness Mühendisliğinin Kavramsal Temelleri ve Alandaki Çalışmalar

Harness mühendisliği terim olarak oldukça yeni olsa da ajan mimarileri ve büyük dil modeli tabanlı sistemlerin çevre bileşenleri üzerine hızla büyüyen bir literatür oluşmuştur. Bu bölümde, alanın kavramsal temellerini oluşturan akademik çalışmalar ile Anthropic, OpenAI, Microsoft ve Google'ın 2023–2026 yılları arasında yayımladığı mühendislik raporları ve kılavuzları incelenmiştir.

Ajan harness'larının kavramsal temeli, Yao ve arkadaşlarının [1] önerdiği ReAct [Reasoning and Acting (Akıl Yürütme ve Eylem)] çerçevesine dayanmaktadır. ReAct, dil modelinin akıl yürütme (reasoning) ile eylemi (action) iç içe geçirdiği, her araç çağrısının sonucunu bir sonraki akıl yürütme adımına girdi yaptığı döngüsel deseni tanımlamış ve bugünkü bütün kodlama ajanlarının temel döngüsünün öncüsü olmuştur. Anthropic'in 2024 sonunda yayımladığı "Building Effective Agents" raporu ise alanın en çok atıf alan mühendislik metinlerinden biri hâline gelmiştir. Bu rapor, önceden tanımlı akışlarla ilerleyen iş akışları (workflows) ile modelin kendi sürecini dinamik olarak yönettiği ajanlar (agents) arasındaki ayrımı netleştirmiştir. Ayrıca, "artırılmış LLM" (augmented LLM) kavramını, yani modelin araçlar, geri çağırma ve hafıza ile donatılmış hâlini, bütün ajan sistemlerinin temel yapı taşı olarak tanımlamış ve karmaşık çerçeveler yerine basit, şeffaf ve birleştirilebilir desenler kullanılmasını önermiştir [2]. Bu temel yaklaşımların ardından literatür, ajanların çalışma döngüsünü destekleyen harness bileşenlerinin; bağlam yönetimi, araç tasarımı ve protokoller, değerlendirme, çoklu ajan mimarileri, güvenlik ve izin modelleri ile kullanıcı odaklı kılavuzlar ekseninde nasıl tasarlanması gerektiğine doğru genişlemiştir.

Bağlam yönetimi konusundaki en sistematik çalışma, Anthropic'in 2025'te yayımladığı "Effective Context Engineering for AI Agents" raporudur. Rapor, bağlamın "sonlu ve getirisi giderek azalan bir kaynak" (raporun özgün ifadesiyle "a finite resource with diminishing marginal returns") olarak ele alınması gerektiğini, bağlam penceresi büyüdükçe modelin bilgiyi geri çağırma isabetinin düştüğünü (context rot) ortaya koymuştur. Buna karşı sıkıştırma (compaction), yapılandırılmış not alma (structured note-taking) ve görevi alt ajanlara bölme olmak üzere üç temel strateji tanımlamıştır [3]. Aynı dönemde istem mühendisliğinden bağlam mühendisliğine geçiş, sektörde geniş kabul görmüştür. Artık "modele ne söylediğiniz değil, bağlam penceresine her turda hangi bilgilerin girdiği" asıl belirleyici olarak tanımlanmıştır. AGENTS.md standardı ise proje düzeyindeki kalıcı bağlamın (derleme komutları, kod standartları, mimari notlar) ajanlar arasında taşınabilir hâle gelmesini sağlayarak bu alandaki en yaygın pratik hâline gelmiştir [4].

Araç katmanı üzerine Anthropic'in "Writing Effective Tools for Agents" (2025) çalışması, araçların deterministik yazılım arayüzlerinden farklı tasarlanması gerektiğini, az sayıda, kapsamlı ve net isimlendirilmiş aracın, çok sayıda parçalı araçtan daha iyi sonuç verdiğini ve araç açıklamalarının da birer istem gibi optimize edilmesi gerektiğini göstermiştir [5]. Kasım 2024'te açık standart olarak yayımlanan MCP (Model Context Protocol), araç ve veri kaynaklarının ajanlara tek tip bir protokolle bağlanmasını sağlamış ve 2025 boyunca OpenAI, Microsoft ve Google tarafından da benimsenerek fiilî endüstri standardı hâline gelmiştir [6]. Editör ile ajan arasındaki iletişimi standartlaştıran ACP (Agent Client Protocol) ve kodun sembolik analizini ajanlara açan LSP (Language Server Protocol) ise harness'ın araç katmanını tamamlayan diğer iki protokol olarak literatürde yerini almıştır [7].

Değerlendirme tarafında alanın standart ölçütü, Jimenez ve arkadaşlarının (2024) geliştirdiği SWE-bench [Software Engineering Benchmark (Yazılım Mühendisliği Kıyaslama Ölçütü)] kıyaslamasıdır. Bu kıyaslama gerçek GitHub sorunlarından türetilen görevlerle modellerin yazılım mühendisliği yeteneğini ölçmektedir [8]. Harness mühendisliği açısından asıl kritik bulgu ise OpenAI'ın 2024'te yayımladığı SWE-bench Verified çalışmasından gelmiştir. Çalışmada aynı modelin farklı harness'lar (scaffold) altında kıyaslama puanının dramatik biçimde değiştiği gösterilmiş, böylece raporlanan "model başarısının" aslında model ile harness'ın ortak başarısı olduğu netleşmiştir [9]. METR [Model Evaluation and Threat Research (Model Değerlendirme ve Tehdit Araştırması)]'in 2025'te yayımladığı kontrollü deney, ölçülen performans ile algılanan performans arasındaki farkı ortaya koymuştur.

16 deneyimli açık kaynak geliştiricisinin 246 gerçek görevde incelendiği bu çalışmada, dönemin yapay zekâ araçlarını kullanan geliştiriciler görevleri %19 daha yavaş tamamlamış, buna rağmen kendilerini %20 hızlanmış hissetmiştir [10]. Bu algı–gerçeklik farkı, ajan kazanımlarının otomatik gelmediğini; doğrulama döngüleri, izin ayarları ve bağlam disiplini gibi harness pratiklerinin bilinçli kurulmasına bağlı olduğunu göstermiştir.

Çoklu ajan mimarileri üzerine Wu ve arkadaşlarının [11] Microsoft bünyesinde geliştirdiği AutoGen çerçevesi, birden fazla ajanın konuşma tabanlı iş birliğiyle görev çözmesini sistematikleştiren ilk yaygın altyapı olmuştur. Anthropic'in 2025'te yayımladığı "How We Built Our Multi-Agent Research System" raporu ise üretim ölçeğinde somut veriler sunmuştur: yönetici–işçi (orchestrator–worker) deseniyle çalışan çoklu ajan sistemi, Anthropic'in şirket içi değerlendirme testlerinde tek ajanlı kuruluma göre %90,2 daha iyi performans göstermiş; performans varyansının yaklaşık %80'inin tek başına token (bağlam) kullanımıyla açıklandığı tespit edilmiştir [12]. Bu bulgu, çoklu ajan mimarisinin özünde bir bağlam ölçekleme tekniği olduğunu; ancak koordinasyon maliyeti ve token tüketimi nedeniyle her göreve uygun olmadığını literatüre kazandırmıştır.

Güvenlik literatüründe ise istem enjeksiyonu (prompt injection), OWASP [Open Worldwide Application Security Project (Açık Dünya Çapında Uygulama Güvenliği Projesi)]'ın 2025 LLM Uygulamaları İçin İlk 10 Risk listesinde birinci sıradaki tehdit olarak tanımlanmıştır [13]. Willison'ın [14] "ölümcül üçlü" (lethal trifecta) çerçevesi, özel veriye erişim, güvenilmeyen içeriğe maruz kalma ve dışarıya veri gönderebilme yeteneklerinin aynı ajanda birleşmesinin yapısal bir sızıntı riski oluşturduğunu göstermiştir. Willison, bu üçlünün ancak harness katmanındaki izin modelleri, sanal alan (sandbox) sınırları ve insan onayı kapılarıyla kırılabileceğini savunmuştur [14]. Model sağlayıcılarının (Anthropic, OpenAI, Google gibi) mühendislik raporları da bu yaklaşımı doğrulamakta; model düzeyindeki eğitimin tek başına yeterli olmadığı, savunmanın harness'ta derinlemesine katmanlanması gerektiği konusunda görüş birliği bulunmaktadır.

Son olarak endüstri kılavuzları, harness mühendisliğini kullanıcı perspektifinden sistematikleştirmiştir. OpenAI'ın "A Practical Guide to Building Agents" kılavuzu model–araç–talimat üçlüsünü ajanın üç temel bileşeni olarak tanımlamış ve güvenlik bariyeri (guardrail) katmanlarını standartlaştırmıştır [15]. Google'ın 2025'te duyurduğu A2A [Agent2Agent (Ajanlar Arası)] protokolü, farklı üreticilerin ajanlarının birbirleriyle çalışabilmesini hedefleyerek harness'lar arası birlikte çalışabilirlik sorununu gündeme taşımıştır; ancak ajan–ajan iletişimine odaklanan bu protokol, tekil kullanıcının harness yapılandırması kapsamının dışında kaldığından bu çalışmada ayrıntılı olarak ele alınmamaktadır [16]. Anthropic'in 2026 Ajansal Kodlama Trendleri Raporu ise kullanıcıların ajan kazanımlarının giderek artan bölümünü model değişikliklerinden değil, AGENTS.md dosyaları, özelleştirilmiş araçlar, yetenek (skill) paketleri ve doğrulama döngüleri gibi harness iyileştirmelerinden elde ettiğini raporlamıştır [17]. Bu bulgular bütünüyle değerlendirildiğinde, harness mühendisliğinin kodlama ajanı kullanıcıları için ayrı ve öğrenilebilir bir disiplin olarak olgunlaştığı görülmektedir.

Alanın bugüne kadarki en kapsamlı sistematik derlemesi ise Li ve arkadaşlarının 2026 tarihli "Agent Harness Engineering: A Survey" çalışmasıdır [18]. Çalışma, harness'ı modelden bağımsız bir sistem katmanı olarak ele almakta ve harness sorumluluklarını yedi katmanda toplayan ETCLOVG taksonomisini önermektedir: yürütme ortamı (Execution), araç arayüzü (Tooling), bağlam yönetimi (Context), yaşam döngüsü ve orkestrasyon (Lifecycle), gözlemlenebilirlik (Observability), doğrulama (Verification) ve yönetişim (Governance). Bu taksonomi, çalışmanın ilerleyen bölümlerindeki incelemede katmanların eksiksiz kapsanmasını gözeten bir kapsam kontrol listesi olarak kullanılacaktır.

# Harness Kavramı ve Tarihsel Gelişimi

"Harness" kelimesinin Türkçe karşılığı "koşum takımı"dır. Yazılım dünyasındaki kökeni ise test altyapılarındaki "test harness" kavramına dayanmaktadır. Test harness, test edilen birimin etrafını saran, ona girdileri veren, çıktılarını yakalayan ve sonucu değerlendiren çerçeve olarak tanımlanabilir. Ajan bağlamında harness da tam olarak bunu yapar. Büyük dil modelinin etrafını sararak ona araçlarını, bağlamını, çalışma ortamını ve güvenlik sınırlarını sağlar. Modelin ürettiği araç çağrılarını gerçek dünyada yürütür ve sonuçlarını tekrar modele taşır. Bir benzetme yapmak gerekirse, model güçlü bir motorsa, harness bu motoru yola çıkaran otomobilin geri kalanıdır: şasi, vites kutusu, direksiyon ve frenler. Motor ne kadar güçlü olursa olsun, aracın yol tutuşunu bu parçalar belirler. Claude Code, Codex, Gemini CLI, Copilot ve OpenCode gibi ürünler benzer modeller üzerine kurulsa da araç kullanımı, bağlam yönetimi ve izin mekanizmaları gibi harness bileşenlerinde farklılaşmaktadır.

## Model ve Ajan Arasındaki Fark

Büyük dil modeli, özünde durumsuz bir metin üreticisidir: kendisine verilen bağlam penceresini okur ve sıradaki token'ları üretir. Modelin dosya sistemi yoktur, terminali yoktur, hafızası yoktur; bir önceki isteği hatırlamaz. Ajan ise bu durumsuz üreticinin bir döngü, bir araç seti ve bir durum yönetimi ile sarmalanmış hâlidir. Anthropic'in tanımıyla ajan, "kendi süreçlerini ve araç kullanımını dinamik olarak yöneten, görevi nasıl tamamlayacağı üzerindeki kontrolü elinde tutan" sistemdir [2]. Bu tanımın pratikteki karşılığı şu formüldür: ajan = model + harness. Model akıl yürütmeyi sağlar; harness ise bu akıl yürütmeyi gerçek bir bilgisayar üzerinde eyleme dönüştürür.

Bu ayrımın kullanıcı için en önemli pratik sonucu, bir performans sorunuyla karşılaşıldığında sorunun modelden mi yoksa harness'tan mı kaynaklandığının doğru teşhis edilebilmesidir. Önceki bölümde aktarılan SWE-bench Verified bulgusu, yani aynı modelin puanının iskele (scaffold) değiştiğinde ciddi şekilde değişmesi, bu teşhisin nicel temelidir [9]. Buradaki "iskele" (scaffold), harness ile sıkça birbirinin yerine kullanılsa da aralarında önemli bir kavramsal ayrım vardır. Scaffold daha eski ve daha geniş bir terimdir: istem şablonları, çıktı ayrıştırma, çağrı zincirleme, araç tanımları ve hafıza yönetimi dâhil, modeli daha yetenekli kılmak için etrafına örülen her türlü destek yapısını kapsar. Harness ise daha yeni ve daha dar bir terimdir. Modele gözlemleri besleyen, araç çağrılarını gerçek ortamda yürüten, sonuçları bağlama geri taşıyan çalışma zamanı döngüsünü ifade eder. Bu yürütme katmanı, modelden gelen tamamlanma sinyaline göre döngüyü sonlandırarak modeli ajana dönüştürür ("ajan çatısı" da bu katmanı adlandırır). Bu çerçevede her ajan harness'ı bir scaffold'dur; ancak her scaffold bir harness değildir. Bir ajan görevde başarısız olduğunda bunun sebebi modelin "yetersizliği" olabileceği gibi, eksik bir araç, taşan bir bağlam penceresi, kötü yazılmış bir sistem istemi ya da doğrulama döngüsünün olmaması da olabilir. Harness mühendisliği bilgisi olmayan kullanıcı bu ayrımı yapamaz ve çözümü yanlış yerde (genellikle model değiştirmekte) arar.

## Harness'ın Doğuşu ve Dönüşümü

Bugünkü harness'ların kavramsal atası, Yao ve arkadaşlarının 2022 sonunda önerip 2023'te yayımladığı ReAct desenidir. Bu desende model önce düşünür, sonra bir eylem seçer, eylemin gözlemi bağlama eklenir ve döngü devam eder [1]. 2023'te AutoGPT ve BabyAGI gibi projeler bu deseni popülerleştirmiş, ancak dönemin modelleri ve ilkel harness'ları ile pratik değer üretmekte zorlanmıştır. Bağlam pencereleri taşmakta, ajanlar sonsuz döngülere girmekte ve hatalarını fark edecek geri bildirim mekanizmaları bulunmamaktaydı. Bu dönemin en önemli mirası, başarısızlıkların model kadar harness kaynaklı olduğunun ilk kez görünür hâle gelmesidir.

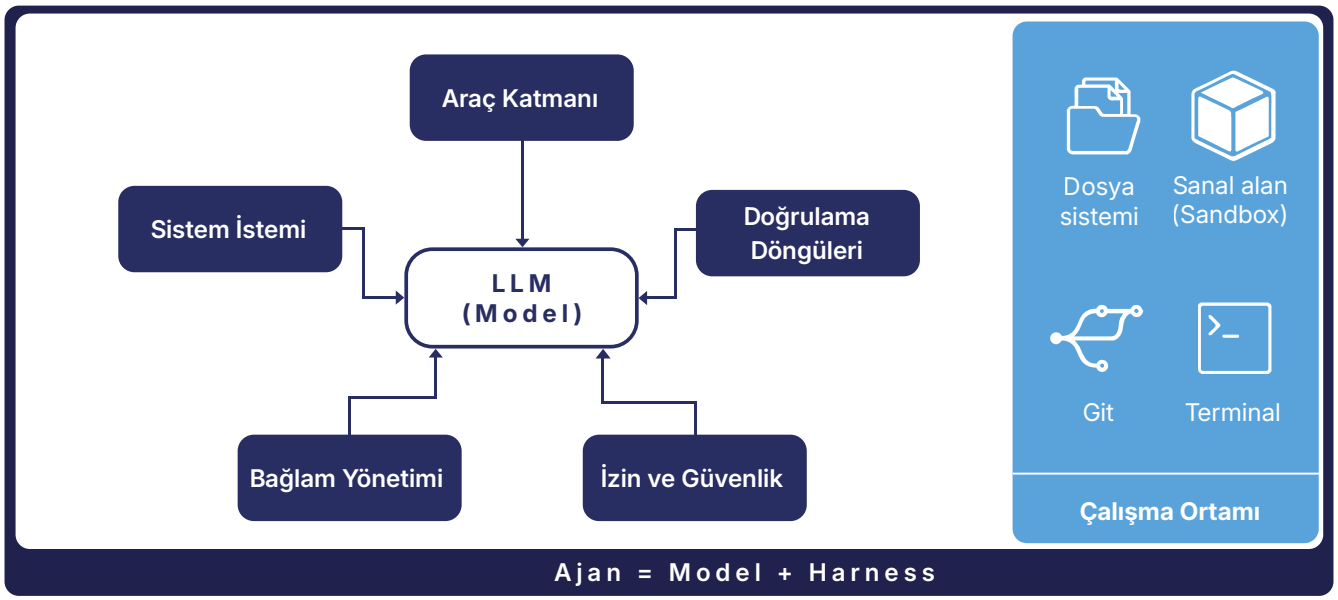
2024 sonunda Anthropic'in "Building Effective Agents" raporu, karmaşık çerçeveler yerine basit ve şeffaf desenleri savunarak alana mühendislik disiplini kazandırmıştır [2]. Asıl kırılma ise 2025'te terminal tabanlı kodlama ajanlarının (Claude Code, Codex CLI, Gemini CLI, OpenCode) yaygınlaşmasıyla yaşanmıştır: bu araçlar, dosya okuma/yazma, komut çalıştırma ve test etme araçlarını tek bir tutarlı harness'ta birleştirmiş ve ajanı geliştiricinin gerçek çalışma ortamına taşımıştır. 2025–2026 döneminde MCP, AGENTS.md ve ACP gibi standartların benimsenmesiyle harness bileşenleri tek tek ürünlerden ayrışarak taşınabilir hâle gelmiştir. Böylece kullanıcıların harness'ı kendi ihtiyaçlarına göre yapılandırması hem mümkün hem de gerekli olmuştur ([4], [6]).

## İstem Mühendisliğinden Harness Mühendisliğine

İstem mühendisliği (prompt engineering), tek bir etkileşimin çıktısını iyileştirmeye odaklanır. Amaç doğru talimatı, doğru örnekleri ve doğru formatı bulmaktır. Ajanlarda ise tek bir etkileşim yoktur; onlarca, bazen yüzlerce tur süren bir döngü vardır. Her turda bağlam penceresine nelerin gireceği ise model ile harness'ın iş birliğiyle belirlenir. Model ihtiyaç duyduğu bilgiyi araç çağrılarıyla talep eder, harness bu çağrıları yürütüp sonuçlarını pencereye yerleştirir. Anthropic bu geçişi "bağlam mühendisliği" (context engineering) kavramıyla tarif etmektedir: artık "İstemi nasıl yazmalıyım?" sorusunun yerini "Her turda modelin sınırlı bağlam bütçesine hangi bilgilerin girmesi en doğrusudur?" sorusu almaktadır [3]. Harness mühendisliği bu perspektifin bir üst kümesidir; bağlamın yanında araçları, doğrulama döngülerini, izin sınırlarını ve çoklu ajan koordinasyonunu da kapsar. Kullanıcı açısından pratik fark, istem mühendisliğinin her görevde tekrar yapılmasıdır. Harness mühendisliği ise bir kez doğru kurulduğunda bütün görevlerin kalitesini kalıcı olarak yükseltir.

# Harness'ın Anatomisi

Bir kodlama ajanı harness'ı, birbirini tamamlayan dört ana bileşenden oluşur: döngüyü yöneten çekirdek (ajan döngüsü), modelin davranışını tanımlayan sistem istemi, modelin dünyaya dokunmasını sağlayan araç katmanı ve bütün bunların üzerinde çalıştığı ortam. Şekil 1'in gösterdiği gibi bu çekirdek yapı; modelin bağlam penceresini besleyen bağlam yönetimi, üretilen işi sınayan doğrulama döngüleri ve ajanın eylemlerini sınırlayan izin ve güvenlik katmanıyla tamamlanır. Bileşenlerin her biri, çıktı kalitesini tek başına belirleyebilecek kadar kritiktir. Bu alt bölümde dört ana bileşen sırasıyla incelenecek; bağlam yönetimi, doğrulama döngüleri ile izin ve güvenlik konuları ise kendilerine ayrılan bölümlerde ayrıntılı olarak ele alınacaktır.

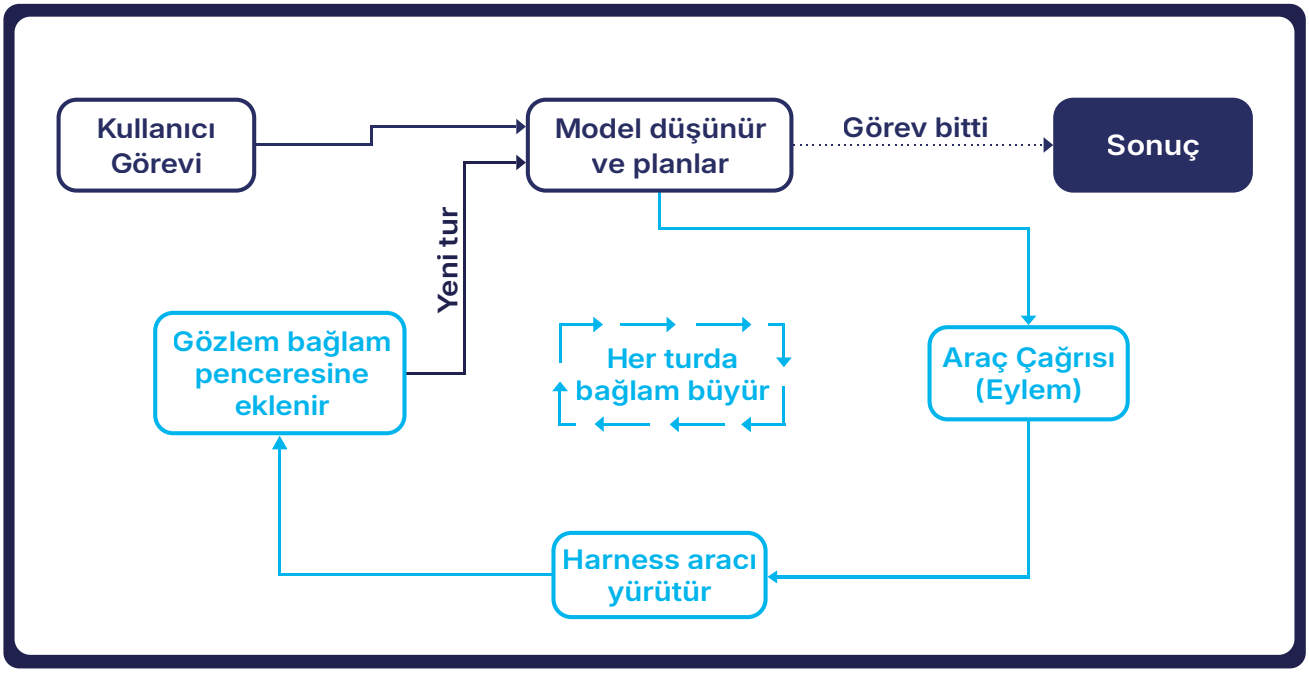



*Şekil 1. Harness'ın Anatomisi*

## Ajan Döngüsü

Ajan döngüsü, harness'ın kalbidir ve aslında oldukça basittir: bağlam topla, eylem üret, eylemi yürüt, sonucu bağlama ekle ve görev bitene kadar tekrarla. Model her turda mevcut bağlam penceresini okur veya bir araç çağrısı üretir ya da görevin bittiğini bildirir. Harness, bu döngü kapsamında bir turda kaç araç çağrısına izin verileceği, toplam tur ve token bütçesinin ne olacağı ve hata alan bir araç çağrısının modele nasıl raporlanacağı gibi kritik kararları vermektedir. Döngünün ne zaman sonlanacağına ise esasen model karar verir: model görevin bittiğini bildirdiğinde harness bu dur sinyaline uyarak döngüyü sonlandırır; bütçe aşımı ya da kullanıcı onayı gereken durumlarda ise döngüyü kesen taraf harness'tır. Anthropic'in mühendislik yazıları bu döngüyü "bağlam topla → eylem al → işi doğrula → tekrarla" şeklinde formüle etmekte (Şekil 2) ve doğrulama adımının döngünün en sık ihmal edilen halkası olduğunun altını çizmektedir ([2], [3]).

Kullanıcı açısından döngünün en önemli özelliği şeffaflıktır. İyi bir harness, hangi turda hangi aracın çağrıldığını, ne sonuç döndüğünü ve modelin neye karar verdiğini görünür kılar. Bu görünürlük olmadan kullanıcı, ajan yanlış bir yöne saptığında bunu ancak iş bittikten sonra fark eder. Görünürlük varsa döngüyü erken kesip yönlendirme yapabilir. Bu yüzden terminal tabanlı ajanların çoğu her araç çağrısını ve özetini akış hâlinde ekrana yazar.




*Şekil 2. Ajan Döngüsü*

## Sistem İstemi

Sistem istemi, harness'ın modele her turda verdiği sabit talimat katmanıdır. Ajanın kimliği, hangi araçları nasıl kullanacağı, kod yazarken uyacağı kurallar, güvenlik sınırları ve çıktı formatı burada tanımlanır. Kullanıcıların çoğu sistem istemini hiç görmez; oysa ajanın "kişiliği" olarak algılanan davranışların neredeyse tamamı bu katmandan gelir. Anthropic, sistem istemi tasarımında "Goldilocks" dengesini (ne aşırı katı ne aşırı soyut, dengeli bir orta nokta) önermektedir. Her senaryoyu ayrı bir kurala bağlamaya çalışan aşırı katı istemler kırılgan davranışlara yol açarken, fazla soyut istemler modele yeterli yönlendirme sağlamaz. Doğru nokta, modele sezgi kazandıracak kadar somut fakat esneklik bırakacak kadar sade talimatlardır [3].

Çoğu ürün, kullanıcıya sistem istemini tamamen değiştirmek yerine kalıcı proje talimatları (AGENTS.md), oturum başı ek talimatlar, ajanın yanıt biçimini ve üslubunu belirleyen çıktı stilleri (output styles) ve özelleştirilmiş ajan tanımları aracılığıyla sisteme ek talimatlar verme imkânı tanımaktadır. Bu ekleme noktalarının bilinçli kullanımı, sistem istemini yeniden yazmaya çalışmaktan hem daha güvenli hem daha etkilidir; kodlama ajanının kendi istemi, araç tanımlarıyla ve döngü mantığıyla birlikte test edilmiş bir bütündür. Bununla birlikte daha ileri kullanım senaryolarında birçok harness, sistem isteminin değiştirilmesine de imkân tanır; bu esneklik, ürünle birlikte doğrulanmış bütünlüğün bozulma riskini beraberinde getirmektedir.

## Araç Katmanı

Araçlar, modelin dünyaya uzanan elleridir. Dosya okuma ve yazma, dizin listeleme, metin arama, terminal komutu çalıştırma, test çalıştırma ve web erişimi gibi yetenekler araç olarak tanımlanır. Model bir araç çağrısını yapılandırılmış bir formatta üretir; harness bu çağrıyı gerçek sistemde yürütür ve sonucunu bir sonraki tura girdi olarak döndürür. Bu mimarinin doğal bir sonucu olarak model, yalnızca araçlarının izin verdiği kadar görebilir ve yapabilir. Kod tabanını arayamayan bir ajan dosyaları tek tek okuyarak bağlamını doldurur. Test çalıştıramayan bir ajan yazdığı kodun doğruluğunu ancak “tahmin” edebilir.

## Çalışma Ortamı ve Yürütücü

Harness’ın son bileşeni, ajanın eylemlerini gerçekleştirdiği dosya sistemi, terminal, Git deposu, derleyiciler ve test altyapısından oluşan çalışma ortamıdır. Bu ortamın iki temel kurulum biçimi vardır. Yerel kurulumda ajan, geliştiricinin kendi makinesinde ve gerçek çalışma kopyası üzerinde çalışır. Hızlıdır ve geliştiricinin bütün araçlarına erişir, fakat hatalı bir komutun gerçek zarar verme riski vardır. Yalıtılmış kurulumda ise ajan bir container veya VM [Virtual Machine (Sanal Makine)] içinde çalışır. Bulut tabanlı ajanların ve CI/CD [Continuous Integration / Continuous Deployment (Sürekli Bütünleştirme / Sürekli Dağıtım)] üzerinde çalışan arka plan ajanlarının standardı budur. OpenAI’ın Codex’i ve benzeri bulut ajanları, her görevi ayrı bir yalıtılmış ortamda çalıştırarak hem güvenliği hem de paralelleştirmeyi sağlamaktadır [15].

## Uzak ve Bulut Tabanlı Ajanlar

Kodlama ajanlarının çalışma ortamı giderek geliştiricinin yerel makinesiyle sınırlı olmaktan çıkmaktadır. Yerel ajan, geliştiricinin bilgisayarındaki dosya sistemi, terminal ve çalışma kopyası üzerinde işlem yapar. Uzak veya bulut tabanlı ajan ise görevi ayrı bir yürütme ortamında çalıştırır; kod deposunu bu ortama alır, gerekli bağımlılıkları kurar, değişiklikleri yapar, testleri çalıştırır ve sonunda kullanıcıya bir çıktı veya değişiklik önerisi sunar. Bu modelde geliştirici, ajanı kendi makinesinde sürekli açık tutmak yerine belirli bir işi uzak bir yürütücüye devreder.

Bu yaklaşımın en önemli katkısı yalıtım ve paralelleştirmedir. Her görev ayrı bir kapsayıcı, sanal makine veya benzeri bir çalışma alanında yürütüldüğünde, hatalı komutların kullanıcının gerçek çalışma ortamına zarar verme riski azalır. Aynı zamanda birden fazla görev, birbirini etkilemeden paralel biçimde çalıştırılabilir. OpenAI Codex’in bulut ortamı bu yapıya örnektir: ajan ayrı bir bulut ortamında kodu okuyabilir, düzenleyebilir, komut çalıştırabilir, testleri deneyebilir ve sonunda kullanıcıya değişiklikleri pull request akışıyla sunabilir [15]. OpenAI belgelerinde Codex’in varsayılan bir konteyner imajı içinde çalıştığı, gerektiğinde çalışma zamanı sürümlerinin sabitlenebildiği ve AGENTS.md dosyasındaki proje talimatlarını kullanabildiği belirtilmektedir [15].

Uzak ajan yaklaşımı yalnızca büyük sağlayıcıların kapalı sistemleriyle sınırlı değildir; OpenCode gibi açık kaynak ajanların bulut tabanlı çalışma ortamlarında çalıştırılmasına yönelik girişimler, kodlama ajanlarının yalnızca geliştiricinin terminalinde çalışan araçlar olmaktan çıkıp daha taşınabilir ajan servislerine dönüşebileceğine işaret etmektedir.

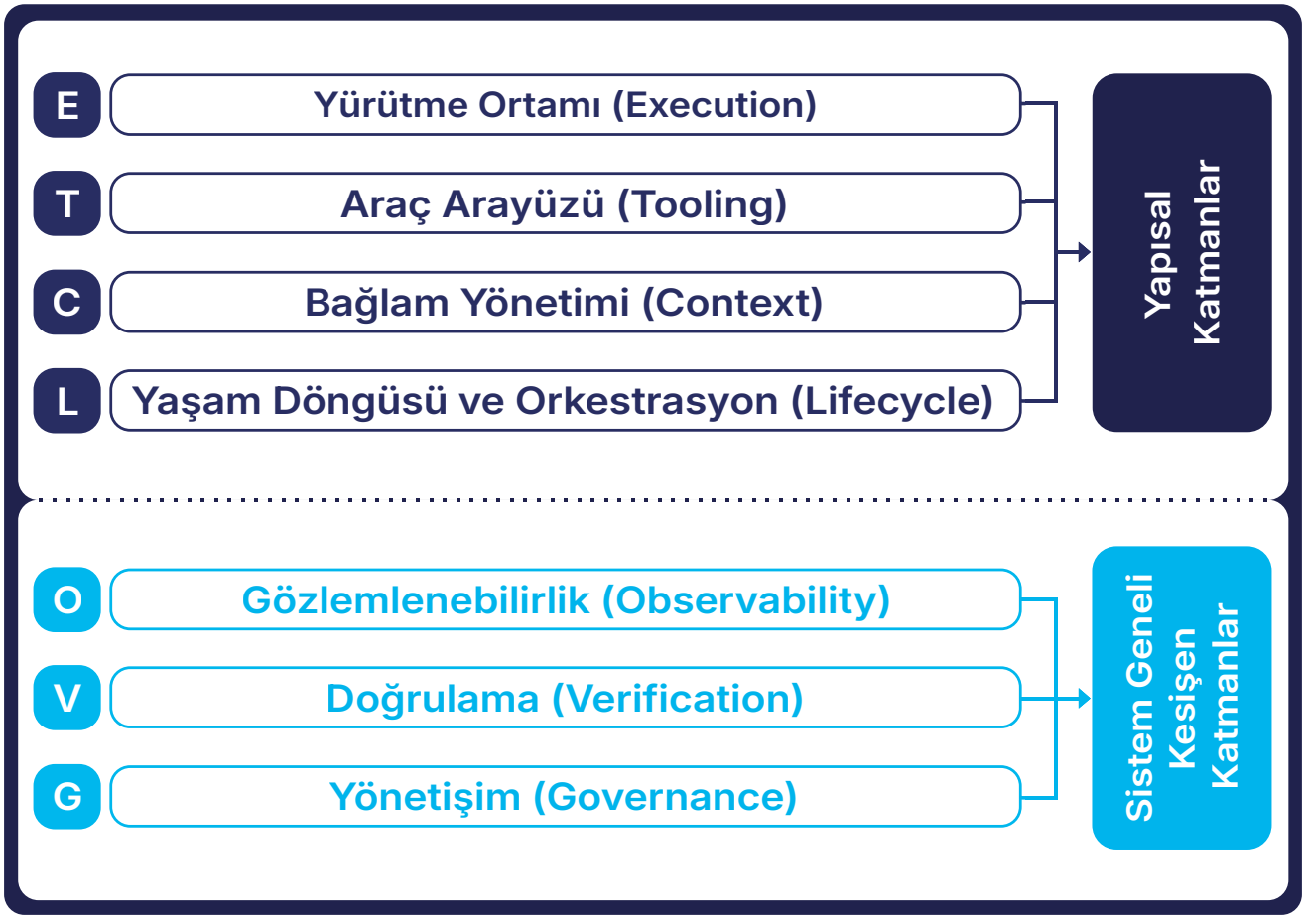
Bununla birlikte uzak ajan modeli yeni riskler ve tasarım soruları da doğurur. Kod deposunun hangi ortama kopyalandığı, bağımlılıkların nereden indirildiği, gizli bilgilerin ajana nasıl verildiği, ağ erişiminin ne kadar açık olduğu ve üretilen değişikliklerin nasıl denetlendiği açıkça belirlenmelidir. Yerel makinede çalışan ajan için risk daha çok dosya sistemi ve komut çalıştırma yetkileridir; uzak ajan için buna depo kopyası, kimlik bilgileri, ağ politikaları ve kalıcı artefaktlar da eklenir. Bu nedenle bulut ajanları, “Güvenlik ve İzin Modeli” bölümünde ele alınan izin modelleriyle ve “Değerlendirme ve Gözlemlenebilirlik” bölümünde ele alınan izleme, telemetri ve maliyet mekanizmalarıyla birlikte değerlendirilmelidir.



## Katmanların Bütüncül Görünümü: ETCLOVG Taksonomisi

Buraya kadar tarif edilen bileşenler, literatürde ETCLOVG taksonomisi ile sistematik bir çerçeveye oturtulmuştur [18] (bkz. Şekil 3). Taksonomi, harness sorumluluklarını yedi katmana ayırır: yürütme ortamı (E) sanal alan ve izolasyonu; araç arayüzü (T) araçları ve entegrasyon protokollerini; bağlam yönetimi (C) bağlam penceresi, sıkıştırma ve hafızayı; yaşam döngüsü ve orkestrasyon (L) ajan döngüsünü, durum yönetimini ve çoklu ajan koordinasyonunu; gözlemlenebilirlik (O) iz kayıtları ile maliyet takibini; doğrulama (V) test, değerlendirme ve geri bildirim döngülerini; yönetişim (G) ise izin modelleri, denetim ve insan gözetimini kapsar. Bu çalışmanın yapısı da bu katmanlarla örtüşmektedir: E katmanı "Çalışma Ortamı ve Yürütücü" ile "Sanal Alan (Sandbox) Mimarileri", T katmanı "Araç Katmanı" ile "Araç Tasarımı ve Protokoller", C katmanı "Bağlam Mühendisliği", L katmanı "Ajan Döngüsü" ile "Çoklu Ajan Mimarileri", V katmanı "Geri Bildirim Döngüleri ve Doğrulama" ile "Değerlendirme (Evals)", O katmanı "İzleme, Telemetri ve Maliyet", G katmanı ise "Güvenlik ve İzin Modeli" ile "İzin ve Otomasyon Seviyesinin Ayarlanması" başlıklarında ele alınmaktadır.

Taksonominin kullanıcı açısından değeri, bir kontrol listesi işlevi görmesidir: Bir harness kurulumu değerlendirilirken yedi katmanın her biri için "Bu katman kim tarafından, nasıl yapılandırılıyor?" sorusu sorulabilir. Derlemenin ekosistem bulgusu da bu listeyi anlamlı kılar. Yürütme, araç ve yaşam döngüsü katmanları için ekosistemde zengin ve yerleşik çözümler bulunurken, gözlemlenebilirlik ve yönetişim katmanlarına yönelik hazır çözümler henüz sınırlı sayıdadır [18]. Yani kullanıcıların en az hazır destek bulacağı, dolayısıyla en bilinçli olması gereken katmanlar tam da ölçüm ve izin katmanlarıdır.




*Şekil 3. Katmanların Bütüncül Görünümü: ETCLOVG Taksonomisi [18]*

# Bağlam Mühendisliği

Bağlam penceresi, ajanın çalışma belleğidir ve harness'ın yönettiği en kıt kaynaktır. Sistem istemi, araç tanımları, konuşma geçmişi, okunan dosyalar ve araç çıktılarının tamamı bu pencereyi paylaşır. Anthropic'in tespitiyle bağlam, "azalan marjinal getirili sonlu bir kaynaktır" [3]. Pencere doldukça modelin içindeki bilgiyi isabetle geri çağırma yeteneği azalmaktadır. Literatürde "context rot" (bağlam çürümesi) olarak anılan bu olgu, uzun oturumlarda ajan kalitesinin sessizce bozulmasının başlıca sebebidir [3]. Bu yüzden modern harness'lar, bağlamı büyütmeye değil, her turda pencereye giren bilgiyi azaltıp isabetli hâle getirmeye yatırım yapmaktadır.

## Tam Zamanında Bağlam ve Ajansal Arama

İlk nesil yaklaşımlar RAG [Retrieval-Augmented Generation (Erişim Destekli Üretim)] yöntemine dayanıyordu: görevle ilgili olabileceği tahmin edilen içerik parçaları, anlamsal arama ile daha görev başlamadan bulunup bağlam penceresine yükleniyordu. Güncel harness'lar ise bu ön yükleme yaklaşımını terk etmiştir: ajan, hangi bilgiye ihtiyaç duyarsa onu görev sırasında o anda getirir. Bu yaklaşım "tam zamanında" (just-in-time) bağlam stratejisi olarak adlandırılır. Ajan, dosya yolları ve arama araçları gibi hafif referanslar üzerinden gerektiğinde gerektiği kadar veri yükler [3]. Kodlama ajanlarında bunun karşılığı ajansal aramadır. Ajan önce dizin yapısını listelemekte, sonra anahtar kelime araması yapmakta, aramada öne çıkan dosyaların yalnızca ilgili bölümlerini okumaktadır. Bu davranış insan geliştiricinin kod tabanında gezinmesine benzer ve bağlamı yalnızca gerçekten kullanılan bilgiyle doldurur. Kullanıcının buradaki görevi, kod tabanını aranabilir kılmaktır. Anlamlı dosya ve dizin isimleri, tutarlı adlandırma ve iyi bölümlenmiş modüller, ajansal aramanın isabetini doğrudan artırır.

## Sıkıştırma (Compaction) ve Yapılandırılmış Not Alma (Structured Note-Taking)

Pencere dolmaya yaklaştığında harness'ın iki temel aracı vardır. Sıkıştırma (Compaction), konuşma geçmişinin modele özetletilerek yerine bu özetin konmasıdır. Kritik kararlar, çözülmemiş hatalar ve uygulama detayları korunurken, artık gereksiz araç çıktıları atılır [3]. Sıkıştırma kayıplıdır; kötü zamanlanmış bir sıkıştırma, ajanın az önce verdiği bir sözü ya da yarım kalan bir işi unutmasına yol açabilir. Bu nedenle sıkıştırma işleminin planlı biçimde gerçekleştirilmesi gerekmektedir. Görevlerin bağlam taşmayacak boyutlarda parçalara bölünmesi ve önemli ara kararların kalıcı bir yere yazdırılması önerilmektedir. İşte ikinci araç olan yapılandırılmış not alma (structured note-taking) bu ihtiyacı karşılar. Ajan, bağlam penceresinin dışında tutulan bir dosyaya (yapılacaklar listesi, ilerleme notları) düzenli olarak yazar ve gerektiğinde bu notları geri okur. Bu teknik, bellek sıfırlansa bile görev sürekliliğini korur [3].



## Kalıcı Bağlam: AGENTS.md ve Hafıza

Oturum içi bağlamın ötesinde, projeye kalıcı bağlam sağlayan katman vardır. Bunun endüstri standardı hâline gelen biçimi, depo köküne konan AGENTS.md dosyasıdır (Claude Code'da CLAUDE.md; OpenCode ve Codex CLI bu dosyayı doğrudan destekler). Derleme ve test komutları, kod standartları, mimari kararlar, dokunulmaması gereken alanlar ve sık yapılan hatalar burada tarif edilir. Harness bu dosyayı her oturumun başında sistem istemine ekler [4]. Etkili bir AGENTS.md dosyasının kısa ve güncel tutulması önerilmektedir; gereğinden fazla ayrıntı içeren bir talimat dosyası ise bağlam bütçesini tüketmekte ve modelin dikkatini dağıtmaktadır. Bu dengeyi kurmanın etkili bir yolu, AGENTS.md'yi çekirdek bir yönlendirme dosyası olarak tutup ayrıntıyı dışarıya taşımaktır: dosyanın kendisi kısa kalır ve yalnızca doğrudan ajana yönelik talimatları (derleme komutları, temel kod kuralları, dokunulmaması gereken alanlar) barındırırken, mimari kararların ve uzun kılavuzların ele alındığı daha kapsamlı dokümanlara referans verir. Böylece ajan her oturumda yalnızca çekirdek talimatları yükler, ayrıntılı belgeyi ise gerektiğinde ajansal aramayla getirir; bağlam bütçesi korunurken proje bilgisi erişilebilir kalır. Buna ek olarak bazı harness'lar oturumlar arası hafıza mekanizmaları sunar: ajanın önceki oturumlardan öğrendiklerini (proje kararları, kullanıcı tercihleri) bir hafıza dizininde saklaması ve yeni oturumda geri yüklemesi. Ajana aynı şeyin ikinci kez açıklanması gerekiyorsa, o açıklama AGENTS.md'ye ya da hafızaya yazılmalıdır.

# Araç Tasarımı ve Protokoller

Araç katmanı, harness'ın en doğrudan yapılandırılabilir bölümlerinden biridir. Ajan yalnızca metin üreten bir model olarak kaldığında çalışma alanını gözlemleyememekte, dosya okuyamamakta, komut çalıştıramamakta veya dış sistemlerle etkileşime geçememektedir. Bu yetenekler, harness'ın modele sunduğu araçlar aracılığıyla sağlanmaktadır. Dosya okuma, metin arama, terminal komutu çalıştırma, test başlatma, web araması yapma veya bir kurumsal sistemden kayıt getirme gibi işlemler bu katmanda yer almaktadır. Bu noktada "araç" kavramı ile bu araçların harness'a nasıl bağlandığını belirleyen protokolleri ayırmak gerekir. Araç, modelin çağırabileceği somut bir işlevdir. MCP, LSP ve ACP ise farklı bütünleşme problemlerini çözen standartlardır. MCP dış sistemlerden araç, kaynak ve istem şablonu sağlamaya; LSP kodun sembolik yapısını editör ve araçlara açmaya; ACP ise ajan arka ucu ile editör ön yüzü arasındaki iletişimi standartlaştırmaya odaklanır. Bu bölümde önce MCP'nin araç ekosistemindeki yeri açıklanacak, ardından harness içindeki araç kavramı ve MCP araçlarının tasarım ilkeleri ele alınacak; bölümün sonunda ise araç katmanını tamamlayan LSP ve ACP protokolleri incelenecektir.

## Model Context Protocol (MCP)

Kasım 2024'te Anthropic tarafından açık standart olarak yayımlanan Model Context Protocol (MCP), ajan ile dış sistemler arasındaki bağlantıyı standartlaştırmaktadır; kendi belgelerinde yer alan benzetmeyle "yapay zekânın USB-C'si" olarak konumlandırılmaktadır [6]. Bir MCP sunucusu, harness'a üç tür yetenek sunabilir: araçlar, kaynaklar ve istem şablonları. Araçlar modelin çağırabileceği eylemleri; kaynaklar modelin okuyabileceği verileri; istem şablonları ise sık kullanılan iş akışları için hazır kalıpları temsil eder.

Mimari açıdan MCP, istemci-sunucu modeline dayanmaktadır. Harness, istemci olarak bir veya daha fazla MCP sunucusuna bağlanır ve sunucuların sunduğu yetenekleri ajanın kullanımına açar.

Sunucular geliştiricinin makinesinde yerel bir süreç olarak çalışabileceği gibi, ağ üzerinden erişilen uzak bir hizmet olarak da sunulabilir. Bu esneklik, aynı bütünleşmenin kişisel kullanımdan kurumsal dağıtıma kadar farklı ölçeklerde yeniden kullanılmasına imkân tanır.

MCP'nin gücü, benimseyen taraf sayısı arttıkça her katılımcı için değerinin de artması anlamına gelen ağ etkisinden gelir. GitHub, veri tabanları, tasarım araçları ve kurumsal sistemler için hazır MCP sunucularının ortaya çıkması, kullanıcıların harness üreticisinin özel entegrasyon yazmasını beklemeden ajanlarına yeni yetenekler kazandırmasını sağlamaktadır. Örneğin bir ekip, iş takip sistemi (issue tracking system), iç dokümantasyon veya günlük kayıt (log) altyapısı için MCP sunucuları bağlayarak ajanın kurumsal bağlama erişimini genişletebilir.

Bununla birlikte MCP sunucusu eklemek yalnızca yetenek kazandırmaz; yeni bir yetki ve bağlam yüzeyi de oluşturur. Her bağlı sunucunun sunduğu araç açıklamaları ve kaynak tanımları, kullanılmasalar bile ajanın karar alanını genişletebilmektedir. Güvenilmeyen sunucular ise veri sızıntısı ve istem enjeksiyonu riski oluşturabilmektedir. Bu nedenle MCP, yalnızca entegrasyon kolaylığı açısından değil, "Güvenlik ve İzin Modeli" bölümünde ele alınan izin modelleri ve güvenlik sınırlarıyla birlikte değerlendirilmelidir.

Kullanıcı açısından temel kural şudur: MCP, dış sistemleri ajana bağlamanın standart yoludur; ancak her sunucu otomatik olarak etkinleştirilmemelidir. Yalnızca ilgili görevde gerçekten gereken sunucular açılmalı, gereksiz araçlar kapatılmalı ve her MCP sunucusu yeni bir yetki alanı olarak değerlendirilmelidir.

## Araçların Harness İçindeki Rolü

Araç, harness'ın modele sunduğu çağrılabilir işlevdir. Model doğal dilde düşünmekte ve karar vermekte; ancak çalışma alanında eylem gerçekleştirmek için araçlara ihtiyaç duymaktadır. Bir dosyayı okumak, dizinde arama yapmak, terminal komutu çalıştırmak, testleri başlatmak veya dış sistemden bilgi getirmek araç çağrılarıyla yapılır. Bu nedenle araç katmanı, modelin soyut akıl yürütmesini somut geliştirme ortamına bağlayan köprü işlevi görür.

Araçların kaynağı farklı olabilir. Bazı araçlar harness'ın kendi içinde yerleşik olarak gelir: dosya okuma, dosya yazma, terminal, arama ve test çalıştırma gibi. Bazıları editör veya IDE [Integrated Development Environment (Bütünleşik Geliştirme Ortamı)] bütünleşmesinden sağlanır. Bazıları ise MCP sunucuları üzerinden dış sistemlerden gelir. Kullanıcı açısından önemli olan, aracın nereden geldiğinden çok ajanın onu hangi yetkiyle, hangi bağlamda ve hangi sınırlar içinde kullanabildiğidir.

Bu ayrım özellikle önemlidir. "MCP tool" ifadesi, araç kavramının tamamını karşılamamaktadır. MCP üzerinden gelen araçlar, harness'ın modele sunduğu araç havuzunun yalnızca bir bölümüdür. Harness kendi yerleşik araçlarını MCP olmadan da sunabilir. Benzer biçimde LSP, doğrudan bir "araç paketi" olmaktan çok, kodun sembolik bilgisini erişilebilir hâle getiren ayrı bir protokoldür. Dolayısıyla araç katmanı, yalnızca MCP'ye indirgenmemelidir.

İyi tasarlanmış bir harness'ta araçlar hem yetenek hem de sınır olarak düşünülür. Bir aracı eklemek, ajana yeni bir imkân verir; fakat aynı zamanda karar alanını genişletir, bağlam penceresinde yer kaplar ve yanlış araç seçimi ihtimalini artırır. Çok sayıda aracın aynı anda açık olması, ajanın daha akıllı davranacağı anlamına gelmez. Aksine, benzer işlevlere sahip çok sayıda araç modeli kararsız bırakabilir ve gereksiz çağrılara yol açabilir.

Kullanıcı açısından temel kural şudur: araçlar varsayılan olarak çoğaltılmamalı, görev için gerekli en küçük yetenek kümesi etkin tutulmalıdır. Dosya okuma, arama ve test çalıştırma gibi sık kullanılan yerleşik araçlar çoğu kodlama görevi için yeterli olabilir. Dış sistemlere erişim, veri tabanı işlemleri veya kurumsal servis çağrıları ise ancak ilgili iş akışı gerçekten bunu gerektiriyorsa eklenmelidir.

## MCP Araçları için Tasarım İlkeleri

Anthropic'in "Writing Effective Tools for Agents" çalışmasının temel bulgusu, ajanlar için araç tasarımının klasik API [Application Programming Interface (Uygulama Programlama Arayüzü)] tasarımından farklı olduğudur [5]. Deterministik bir sistem her fonksiyonu doğru sırayla çağıracak biçimde programlanır. Ajan ise araçları isimlerinden, açıklamalarından ve o anki görev bağlamından hareketle seçer. Bu nedenle bir aracın teknik olarak doğru çalışması yeterli değildir; model tarafından doğru zamanda ve doğru gerekçeyle seçilebilecek biçimde tasarlanması gerekir.

Bu ilkeler özellikle MCP sunucusu yazarken veya hazır MCP sunucuları seçerken önem kazanır. Birinci ilke, az sayıda ve kapsamlı aracın çok sayıda parçalı araçtan çoğu zaman daha iyi sonuç vermesidir. On beş ayrı uç nokta yerine, kullanıcının gerçek iş akışını tek adımda tamamlayan konsolide bir araç daha verimli olabilir. Örneğin "müşteriyi bul", "kayıtlarını listele" ve "duruma göre filtrele" biçiminde üç ayrı araç yerine, "müşterinin açık destek taleplerini getir" aracı hem daha az karar yükü hem de daha az bağlam tüketimi üretir.

İkinci ilke, araç açıklamalarının birer küçük istem olarak yazılmasıdır. Ajan, aracı yalnızca teknik imzasına göre değil, açıklamada sunulan kullanım amacı ve görev bağlamına göre seçmektedir. Bu nedenle açıklamada aracın ne yaptığı kadar ne zaman kullanılması ve ne zaman kullanılmaması gerektiği de belirtilmelidir. Örneğin "kayıt getirir" gibi genel bir açıklama yerine, "kullanıcının açık destek taleplerini müşteri numarasına göre listeler; geçmiş ve kapanmış talepler için kullanılmamalıdır" gibi kullanım sınırları açıkça belirtilen bir açıklama tercih edilmelidir.

Üçüncü ilke, araç yanıtlarının bağlam bütçesini koruyacak biçimde tasarlanmasıdır. Bir arama aracının binlerce satır ham çıktı döndürmesi, ajanın bağlam penceresini tek çağrıda tüketebilir. İyi araçlar özetlenmiş, sayfalanmış ve gereksiz ayrıntıdan arındırılmış yanıtlar döndürür. Yanıtın her satırı, ajanın karar vermesine katkı sağlamalıdır. Ayrıntı gerekiyorsa ilk çağrı özet döndürmeli, ayrıntı için ayrı bir çağrı yapılmalıdır.

Dördüncü ilke, araçların etki alanının açık olmasıdır. Okuma yapan araçlarla yazma veya silme işlemi yapan araçlar aynı risk düzeyinde değildir. Bu nedenle araç tasarımında yalnızca işlev değil, izin modeli de düşünülmelidir. Salt okunur araçlar daha geniş kullanılabilirken, kalıcı değişiklik yapan araçlar onay, kayıt ve geri alma mekanizmalarıyla birlikte ele alınmalıdır.

Kullanıcı bu ilkeleri iki aşamada uygulamaktadır: kendi MCP sunucularını yazarken ve hazır sunucuları seçerken. Çok sayıda araç barındıran bir MCP sunucusunu olduğu gibi bağlamak çoğu zaman ajanı akıllandırmak yerine yavaşlatır. İhtiyaç duyulan birkaç aracı seçip gerisini kapatmak neredeyse her zaman daha iyi sonuç verir. Kullanıcı açısından temel kural şudur: MCP aracı, teknik uç noktanın birebir dışa açılması değil, ajanın anlayabileceği ve güvenle kullanabileceği bir iş akışı birimi olarak tasarlanmalıdır.

## Language Server Protocol (LSP)

Microsoft'un editörler için geliştirdiği LSP, kodun sembolik analizini (tanıma gitme, referans bulma, tip hataları, yeniden adlandırma) dil sunucuları üzerinden standartlaştırır [19]. Ajanlar için önemi, düz metin aramasının kodun anlamsal yapısını görememesinde yatar. LSP entegrasyonu olan bir harness ise "bu fonksiyonu kim çağırıyor?" sorusunu kesin olarak yanıtlayabilir, yazılan kodun tip hatalarını daha dosya kaydedilirken ajana geri bildirir. Bu, hem bağlam tüketimini düşürür (yüzlerce satır okumak yerine tek sembol sorgusu) hem de geri bildirim döngüsünü hızlandırır. Kullanıcı, kullandığı harness LSP entegrasyonu sunuyorsa bunu etkinleştirerek ve projede kullanılan diller için ilgili dil sunucularını kurarak, ajanın kodu düz metin olarak değil anlamsal yapısıyla birlikte görmesini sağlayabilir.

## Agent Client Protocol (ACP)

Zed ekibinin öncülük ettiği Agent Client Protocol (ACP), ajan arka ucu ile editör/istemci ön yüzü arasındaki iletişimi standartlaştırmayı hedeflemektedir. Language Server Protocol (LSP) editör–dil sunucusu ilişkisini nasıl genelleştirdiyse, ACP de benzer biçimde editör–ajan ilişkisini genelleştirmeye çalışır [7]. Temel fikir, her editörün her ajan için ayrı bir entegrasyon geliştirmesine gerek kalmamasıdır. ACP konuşan bir editör, aynı standart arayüz üzerinden farklı ajanları çalıştırabilir; ACP konuşan bir ajan da farklı editörlere özel ayrı eklentiler yazılmadan kullanılabilir.

Bunun kullanıcıya yansıması sağlayıcı bağımsızlığıdır. Kullanıcı, mevcut editör arayüzünü koruyarak farklı ajanları kullanabilmektedir. Örneğin bir ekip aynı geliştirme ortamı içinde farklı modelleri veya farklı kodlama ajanlarını karşılaştırabilir; arayüz değişmediği için öğrenme maliyeti azalır. Bu yönüyle ACP, ajan ekosistemindeki kilitlenmeyi azaltan ve entegrasyon maliyetini düşüren önemli bir standartlaşma adımıdır. JetBrains'in ACP desteğini, dış ajanların AI Chat içine standart bir arayüzle bağlanabilmesi olarak konumlandırması da bu hedefle uyumludur. Bununla birlikte ACP, ajanın kendi yerel istemcisindeki tüm özelliklerin başka bir editöre eksiksiz taşınacağı anlamına gelmez. Protokol, ortak iletişim zeminini tanımlar; ancak her ajanın kendine özgü ekranları, kısayolları, oturum yönetimi, izin akışları, çıktı gösterimi veya özel araç davranışları bu ortak zemine birebir sığmayabilir. Örneğin OpenCode bir ACP uyumlu editör içinde çalıştırıldığında temel ajan işlevleri kullanılabilir; fakat OpenCode'un kendi arayüzünde sunduğu bazı ayrıntılı deneyimler aynı biçimde erişilebilir olmayabilir. Bu durum ACP'ye özgü bir başarısızlık değil, erken dönem ajan protokollerinin ortak sınırlılığıdır.

Bu sınırlılığın temel nedeni, standartların genellikle en küçük ortak payda etrafında şekillenmesidir. Protokol erken standartlaşırsa farklı araçları hızlıca bir araya getirir; ancak ekosistem henüz olgunlaşmadan sabitlenen arayüz, ajanların yenilikçi veya ürünlere özgü özelliklerini taşımakta yetersiz kalabilir. Bu nedenle ACP kullanımı değerlendirilirken yalnızca "ajan editöre bağlanıyor mu?" sorusu yeterli değildir. "Hangi özellikler protokol üzerinden taşınıyor, hangileri ajanın kendi istemcisine bağlı kalıyor?" sorusu da ayrıca sorulmalıdır.

Kullanıcı açısından temel kural şudur: ACP, ajanlar arasında geçişi kolaylaştıran bir uyumluluk katmanıdır; ancak tam özellik eşdeğerliği garantisi değildir. Standart arayüz, temel kullanım ve karşılaştırma için güçlü bir avantaj sağlar. Buna karşılık, belirli bir ajanın gelişmiş özellikleri yoğun biçimde kullanılıyorsa, o ajanın kendi yerel istemcisi ile ACP üzerinden çalışan sürümü arasında işlevsel farklar oluşabileceği dikkate alınmalıdır.

Bu üç protokolün tamamladığı tabloya, Google'ın 2025'te duyurduğu A2A [Agent2Agent (Ajanlar Arası)] protokolü de eklenebilir [16]. MCP ajanın dış sistemlere, LSP kodun sembolik yapısına, ACP ise editör ön yüzüne bağlanmasını standartlaştırırken; A2A, farklı üreticilerin ajanlarının birbirleriyle görev alışverişi yapabilmesini, başka bir deyişle ajan–ajan iletişimini hedeflemektedir. Kodlama ajanı kullanıcısı açısından A2A henüz günlük iş akışının parçası olmaktan uzaktır; ancak çoklu ajan mimarilerinin farklı sağlayıcıların ajanlarını kapsayacak biçimde genişlemesi durumunda, harness'lar arası birlikte çalışabilirliğin standart katmanı olmaya adaydır. Bu nedenle A2A, bu çalışmada ayrıntılı incelenmemekle birlikte, araç ve protokol katmanının gelişim yönünü gösteren bir standart olarak izlenmelidir.

# Geri Bildirim Döngüleri ve Doğrulama

Ajan döngüsündeki "işi doğrula" adımı, üretilen kodun değerlendirilmesi açısından önemli bir aşamadır. Model, ürettiği kodun çalışıp çalışmadığını kendi başına belirleyememekte; bunu ancak çalışma ortamından aldığı geri bildirimle öğrenmektedir. Harness mühendisliğinin belki de en yüksek getirili yatırımı, bu geri bildirim döngülerini hızlı, güvenilir ve otomatik hâle getirmektir. Geri bildirimi olmayan bir ajan, gözü kapalı kod yazan bir geliştiriciye benzer; ne kadar yetenekli olursa olsun, hatalarını göremediği için düzeltemez.

## Test, Lint ve Mimari Kural Tabanlı Doğrulama

En güçlü geri bildirim kaynağı testlerdir. TDD [Test-Driven Development (Test Odaklı Geliştirme)], kodlama ajanlarıyla birlikte yeni bir anlam kazanmıştır. Önce testlerin yazılması (ya da ajana yazdırılması), ardından ajanın bu testleri geçene kadar kodu yinelemesi, ajana ölçülebilir ve nesnel bir hedef sağlar [20]. Anthropic'in en iyi uygulama önerileri, ajanın hedef üzerinde yinelemeli çalışabildiği bu tür kurulumların tek seferde kod üretmeye göre belirgin biçimde daha başarılı sonuçlar verdiğini vurgulamaktadır. Testlerin yanında derleyici hataları, tip denetimi, lint kuralları ve kod biçimlendiricileri de her yinelemede otomatik olarak çalıştırılabilen düşük maliyetli doğrulama mekanizmalarıdır. Kullanıcının görevi ise bu altyapıyı hazırlamaktır: hızlı çalışan bir test paketi, tek komutla çalışan derleme süreci ve ajanın çağırabileceği, AGENTS.md dosyasında açıkça tanımlanmış doğrulama komutları sağlamak. Doğrulama komutunun on dakika sürdüğü bir projede ajanın yineleme döngüsü fiilen yavaşlar; bu nedenle testlerin çalışma süresi, kodlama ajanlarıyla geliştirilen projelerde doğrudan bir verimlilik ölçütü hâline gelmiştir.

Test ve lint doğrulamalarının bir sonraki aşaması, mimari kuralların da otomatik olarak denetlenmesidir. ArchUnit gibi araçlar, katman bağımlılıkları, paket sınırları, adlandırma kuralları ve döngüsel bağımlılık yasakları gibi mimari kuralların test olarak tanımlanmasına olanak sağlamaktadır. Böylece bu kurallar yalnızca AGENTS.md dosyasında tanımlanan yönergeler olarak kalmamakta; doğrulama mekanizmasının bir parçası hâline gelmektedir. Ajan tarafından üretilen kod derleme ve birim testlerinden geçse bile mimari kuralları ihlal ediyorsa, bu durum otomatik olarak tespit edilir ve ajana açık bir geri bildirim olarak sunulur. Aynı rolü diğer ekosistemlerde benzer araçlar (JavaScript/TypeScript'te dependency-cruiser, Python'da import-linter, .NET'te NetArchTest) üstlenmektedir.

## Plan Modu ve İnsan Onayı Kapıları

Geri bildirimin bir diğer biçimi, eylemden önce alınan insan onayıdır. Modern harness'ların çoğu bir "plan modu" sunar. Bu modda ajan, koda dokunmadan önce görevi analiz eder, bir uygulama planı çıkarır ve kullanıcının onayını bekler. Plan aşamasında yapılan bir dakikalık düzeltme, uygulama aşamasında saatlerce sürecek yanlış yönlenmeyi engeller. Bu yüzden deneyimli kullanıcılar üç beş dakikayı aşacak her iş için önce plan isteme alışkanlığı geliştirmiştir. Benzer şekilde, geri döndürülmesi zor eylemler (dosya silme, git push, veri tabanı değişiklikleri, harici API çağrıları) için harness'lar onay kapıları sunar. Bu kapıların tamamen kapatılması (tam otomatik mod) hız kazandırır fakat hem güvenlik hem kalite riskini büyütür. Doğru ayar, görevin riskine ve ortamın yalıtımına göre kademelendirmedir.

## Doğrulayıcı Ajanlar ve Eleştirmen Desenleri

Üçüncü doğrulama katmanı, işi yapan modelin dışında ikinci bir model gözü kullanmaktır. Yaygın desenlerden biri değerlendirici–iyileştirici (evaluator–optimizer) döngüsüdür. Bir model üretir, ikinci model tanımlı kriterlere göre eleştirir ve üretici bu eleştiriyle yineler [2]. Kodlama pratiğinde bu, temiz bağlamla açılan bir "gözden geçirici" alt ajanın değişiklikleri incelemesi şeklinde uygulanır. Üretici ajanın kendi ürettiği çıktıdaki hataları tespit etmedeki sınırlılığı, temiz bağlamla çalışan eleştirmen ajan için geçerli değildir. Ancak bu desenler token maliyetini katladığından, her görevde değil; kritik, geniş kapsamlı ya da güvenlik hassasiyeti olan değişikliklerde kullanılması önerilir. Nihai gözden geçirmenin insanda kalması ise 2026 itibarıyla bütün üretici kılavuzlarının ortak tavsiyesidir. Ajan çıktısı, dış dünyaya (üretim ortamı, müşteri, ana dal) çıkmadan önce sorumluluğu taşıyabilecek bir insan tarafından anlaşılmış olmalıdır ([15], [17]).

# Güvenlik ve İzin Modeli

Dosya sistemine yazabilen, komut çalıştırabilen ve ağa erişebilen bir yazılım parçası, tanımı gereği bir güvenlik konusudur. Bu yazılımın eylemlerini bir dil modelinin ürettiği durumda konu daha da kritikleşir. Harness'ın güvenlik katmanı üç soruya cevap verir: ajan neyi yapabilir (izin modeli), nerede yapabilir (yalıtım) ve kimin sözünü dinler (istem enjeksiyonuna karşı savunma).

## İzin Modelleri ve Otomasyon Seviyeleri

Modern kodlama ajanları kademeli izin modelleri sunar. En temkinli uçta ajan her eylem için onay ister. Orta kademede okuma ve arama serbest, yazma ve komut çalıştırma onaya tabidir. En serbest kademede ise ajan tam otomatik çalışır. İyi harness'lar bu kademeleri eylem türüne göre daha ayrıntılı biçimde yapılandırır: belirli komutlara kalıcı izin verilebilir (örneğin test çalıştırma), belirli dizinlere erişim yasaklanabilir ve belirli araçlar tümüyle devre dışı bırakılabilir. Kullanıcı açısından temel ilke, otomasyon seviyesini ortamın yalıtımıyla orantılamaktır. Gerçek makinede ve canlı verilerle çalışan bir ajana tam otomasyon vermek yüksek risklidir. Buna karşılık, yalıtılmış bir ortamda ve kolayca silinebilen bir çalışma dalında çalışan ajana daha geniş yetki verilebilir. OpenAI'ın ajan kılavuzu da benzer bir çerçeve önermektedir: yüksek riskli eylemler için insan onayı, düşük riskli eylemler için otomasyon ve her durumda ajanın gerçekleştirdiği işlemlerin sonradan incelenebilmesi için kayıt altına alınması (eylem günlüğü) [15].

İzin modelinin doğru yapılandırılması, iki karşıt riski birden yönetmeyi gerektirir. Bir uçta aşırı serbestlik, diğer uçta izin yorgunluğu vardır: ajan her küçük eylem için onay isterse, kullanıcı bir süre sonra istekleri okumadan onaylamaya başlar ve onay mekanizması gerçek koruma işlevini yitirir. Bu nedenle iyi tasarlanmış harness'lar izni eylemin türüne ve etki alanına göre ayrıştırır. İyi bir pratik, ajanın çalıştığı dizin içinde kalan işlemlere serbestçe izin verirken bu sınırın dışına çıkan işlemleri (dış ağa erişim, çalışma alanı dışındaki dosyalara dokunma) onaya bağlamaktır; böylece sık ve düşük riskli eylemler akışı kesmez, gerçek risk taşıyan eylemler kullanıcının dikkatine sunulur. Aynı ayrıştırma komut düzeyinde de kurulabilir: kullanıcı, geri döndürülmesi zor bir işlemi (örneğin git checkout) tümüyle yasaklarken, denetlenebilir bir işlemi (örneğin bir sürüm kaydı, git commit) yalnızca insan onayıyla yaptıracak özel kurallar tanımlayabilir. Bu ayrıntılı (granular) izin ve onay kuralları, otomasyon hızını korurken riski eylemin gerçek sonucuna göre ölçeklendirmenin en pratik yoludur.

## Sanal Alan (Sandbox) Mimarileri

İzin modelinin tamamlayıcısı, ajanın hata yaptığında verebileceği zararı fiziksel olarak sınırlandıran yalıtımdır. Yaygın kademeler şunlardır: yalnızca çalışma dizinine yazma izni veren dosya sistemi kısıtları; ağ erişimini beyaz listeye indiren ağ kısıtları; ajanı tek kullanımlık bir kapsayıcı ya da VM'de çalıştıran tam yalıtım. Bulut tabanlı arka plan ajanları bu en katı kademeyi standart olarak kullanır. Her görev, deponun bir kopyasıyla açılan taze bir ortamda çalışır, sonuç bir değişiklik önerisi (pull request) olarak dışarı çıkar ve ortam imha edilir [15]. Yerel kullanımda ise git'in kendisi hafif bir güvenlik ağıdır. Ajanın her zaman ayrı bir dalda çalıştırılması ve sık commit yapılması, en kötü senaryoda bile geri dönüşü kolaylaştırır [20]. Bu mimarilerin ortak amacı, "ajan hata yapmasın" gibi gerçekçi olmayan bir hedef yerine, "hata yaptığında zarar sınırlı ve geri alınabilir olsun" hedefini mühendisliğe dönüştürmektir.

## İstem Enjeksiyonu ve Ölümcül Üçlü

Ajan güvenliği açısından önemli tehditlerden biri istem enjeksiyonudur (prompt injection). Bu tehdit modelin, işlediği veri içine gizlenmiş talimatları kullanıcı talimatı sanıp uygulamasıdır. Kodlama ajanları bu tehdide özellikle açıktır, çünkü sürekli olarak güvenilmeyen içerik işlerler: bağımlılık paketlerinin kaynak kodu, iş tanımları, web sayfaları ve hata çıktıları. OWASP, istem enjeksiyonunu LLM uygulamaları için bir numaralı risk olarak sınıflandırmaktadır [13]. Willison'ın "ölümcül üçlü" çerçevesinde riskin yapısı Şekil 4'te sunulmuştur. Özel veriye erişim, güvenilmeyen içeriğe maruz kalma ve dışarıya veri iletebilme yetenekleri aynı ajanda birleştiğinde, verinin sızdırılabileceği bir kanal yapısal olarak açılmış demektir [14]. Bu birleşim, ajanı evin anahtarını taşıyan, kapıya bırakılan her notu okuyup uygulayan ve dışarıya mektup gönderebilen bir kâhyaya dönüştürür; kâhya ne kadar sadık olursa olsun, hırsızın tek yapması gereken doğru notu yazmaktır. Model eğitimi bu saldırıları zorlaştırsa da tek başına yeterli görülmemektedir. Savunma harness'ta katmanlanır: üçlünün en az bir ayağını kırmak (örneğin ağ erişimini kapatmak), güvenilmeyen içerikten gelen talimat benzeri metinleri işaretleyip kullanıcı onayına düşürmek, hassas eylemleri onay kapısına bağlamak ve MCP sunucuları gibi üçüncü taraf bileşenleri kurum düzeyinde denetimden geçirmek. Kullanıcının unutmaması gereken nokta ise ajana verilen her yeni yetkinin ve bağlanan her yeni veri kaynağının aynı zamanda potansiyel bir saldırı yüzeyi olduğudur.




*Şekil 4. Ölümcül Üçlü [14]*

# Çoklu Ajan Mimarileri

Tek ajanın temel darboğazı bağlam penceresidir. Görev büyüdükçe tek pencereye sığmayan bilgi, çıktı kalitesini olumsuz etkileyebilmektedir. Çoklu ajan mimarileri bu darboğazı, işi her biri kendi temiz bağlam penceresine sahip birden fazla ajana bölerek aşmayı hedeflemektedir. Ancak bu mimariler maliyetsiz değildir; koordinasyon karmaşıklığı ve token maliyeti, kazanımı kolayca sıfırlayabilir. Bu bölümde temel desenler ve bunların hangi koşullarda değer ürettiği incelenecektir.

## Alt Ajanlar (Subagents)

En yaygın ve en düşük riskli çoklu ajan biçimi, ana ajanın belirli bir alt görevi temiz bağlamla açılan bir alt ajana devretmesidir. Alt ajan, örneğin "bu hatanın kök nedenini bul" görevini alır. Onlarca dosya okuyup on binlerce token'lık bir araştırma yürütür ve ana ajana yalnızca birkaç yüz token'lık damıtılmış bir özet döndürür. Böylece ana ajanın bağlam penceresi, araştırmanın ham ayrıntısıyla değil sonucuyla dolar. Anthropic bu deseni bağlam mühendisliğinin temel stratejilerinden biri olarak tanımlamaktadır [3]. Harness sistemleri bu deseni hazır alt ajan türleriyle (araştırmacı, gözden geçirici, test yazıcı) ve kullanıcı tanımlı özelleştirilmiş ajanlarla desteklemektedir. Her alt ajana ayrı sistem istemi, ayrı araç seti ve gerekirse ayrı model atanabilir. Ucuz ve hızlı bir modelin keşif işlerine, güçlü bir modelin ise tasarım ve uygulamaya atanması, maliyet–kalite dengesinin yaygın bir optimizasyonudur.

## Orkestrasyon Desenleri

Çoklu ajan mimarisinde en yaygın orkestrasyon deseni, bir yönetici (orchestrator) ajanın görevi alt parçalara bölmesi ve bu parçaları işçi (worker) ajanlara dağıtmasıdır. Anthropic'in çoklu ajanlı araştırma sistemi, bu desenin uygulamada ayrıntılı biçimde belgelenmiş en iyi örneklerinden biridir. Bu mimaride yönetici ajan, kullanıcı isteğini doğrudan tek başına çözmeye çalışmak yerine önce plan çıkarır, araştırılması gereken yönleri belirler, her yön için ayrı işçi ajanlar başlatır ve sonunda gelen bulguları birleştirir. Anthropic'in kendi değerlendirmelerinde bu yönetici–işçi kurulumu, tek ajanlı kuruluma göre %90,2 daha iyi sonuç vermiştir. Aynı analizde performans varyansının yaklaşık %80'inin token kullanımıyla açıklanabildiği görülmüştür [12]. Bu bulgu, çoklu ajan mimarisinin özünde bir bağlam ölçekleme tekniği olduğunu gösterir. Sistem, tek bir bağlam penceresine sığmayacak araştırma alanını birden fazla pencereye dağıtır. Böylece her işçi ajan kendi alt görevinin ayrıntılarına odaklanırken, yönetici ajan genel hedefi ve karar bütünlüğünü korur. Bu yapı özellikle araştırma, karşılaştırmalı analiz ve büyük kod tabanlarında keşif gibi geniş görevlerde etkilidir. AutoGen gibi çerçeveler de konuşma tabanlı iş birliği, rol dağılımı, grup sohbeti ve sıralı konuşma gibi alternatif orkestrasyon desenlerini sistematikleştirmiştir [11]. Ancak çoklu ajan mimarisinin asıl zorluğu işçi ajanları başlatmak değil, onların ürettiği çıktıları uyumlu hâle getirmektir. İki işçi ajan aynı dosyada farklı değişiklikler önerebilir, aynı sınıfın sorumluluğunu farklı yorumlayabilir veya birbirinin varsayımlarını bozan çözümler geliştirebilir. Bu durum özellikle kodlama görevlerinde belirgindir; çünkü kod değişiklikleri çoğu zaman ortak dosyalar, ortak tipler ve ortak testler üzerinden birbirine bağlanır. Bu nedenle çoklu ajan sistemlerinde koordinasyon ve çakışma yönetimi, model kalitesi kadar önemli bir harness mühendisliği problemidir.

Bu problemi azaltmak için kullanılan ilk desen, işçi ajanlara açık sınırlar vermektir. Her işçiye yalnızca hedef değil, çalışabileceği dosya aralığı, değiştirmemesi gereken bileşenler, beklenen çıktı formatı ve durma koşulu da verilmelidir. Örneğin bir işçi yalnızca veri erişim katmanını incelemekle, diğeri yalnızca test eksiklerini bulmakla, bir diğeri ise güvenlik etkilerini değerlendirmekle görevlendirilebilir. Böylece işçiler aynı alan üzerinde bağımsız karar vermek yerine, birbirini tamamlayan sınırlı alanlarda çalışır. Anthropic'in raporunda da işçi ajanlara verilen görev tanımlarının netliğinin, çoklu ajan sisteminin başarısı için belirleyici olduğu vurgulanmaktadır [12].

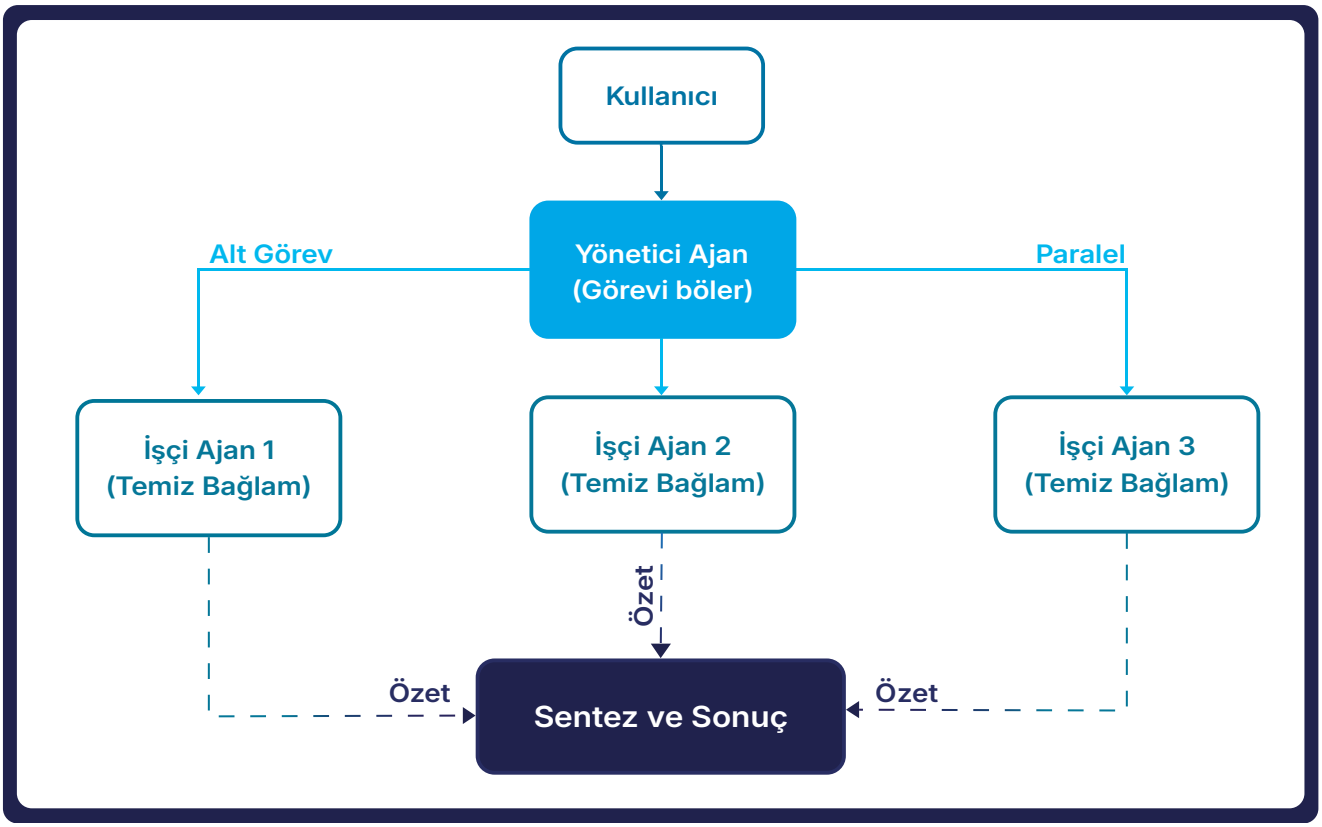
İkinci desen, işçi ajanların doğrudan kodu değiştirmesi yerine önce öneri veya yama taslağı üretmesidir. Bu yaklaşımda alt ajanlar kendi bulgularını, gerekçelerini ve önerilen değişiklikleri yapılandırılmış biçimde yönetici ajana iletir. Nihai değişikliği ise tek bir ana ajan uygular. Böylece çakışan düzenlemeler dosya üzerinde aynı anda işlenmez; önce yönetici tarafından karşılaştırılır. Bu desen, özellikle aynı dosyaya temas etme ihtimali bulunan görevlerde daha güvenlidir.

Üçüncü desen, yönetici ajanın birleştirme ve hakemlik rolünü açıkça üstlenmesidir. Yönetici yalnızca alt ajanlardan gelen çıktıları toplamaz; çelişen önerileri belirler, ortak varsayımları kontrol eder, hangi önerinin uygulanacağını seçer ve gerekirse yeni bir uzlaştırma turu başlatır. Örneğin iki işçi farklı API tasarımları önerdiyse, yönetici bunları tutarlılık, test edilebilirlik ve mevcut mimariyle uyum açısından karşılaştırır. Karar verilemiyorsa görev bir "hakem" ajana veya insan kullanıcıya yükseltilebilir. Böylece çakışma, sessizce koda yansımadan önce görünür hâle getirilir.

Dördüncü desen, paylaşılan çalışma belleği veya durum özeti kullanmaktır. İşçi ajanlar birbirlerinin tüm bağlamını görmek zorunda değildir; ancak üzerinde anlaşılan kararlar, kilit varsayımlar ve kesinleşmiş kısıtlar kısa bir ortak özet olarak tutulabilir. Yönetici ajan bu özeti güncelleyerek sonraki işçilere aktarır. Bu yöntem, işçilerin aynı konuyu yeniden keşfetmesini azaltır ve farklı alt görevlerin aynı mimari kararlara dayanmasını sağlar. AutoGen'in grup sohbeti ve konuşma yöneticisi desenleri de benzer biçimde, birden fazla ajanın ortak bir konuşma akışı içinde sıralanmasını ve mesaj yönlendirmesinin denetlenmesini mümkün kılar [11].

Kodlama görevleri açısından en güvenli uygulama modeli, paralel keşif ile tekil uygulamayı birbirinden ayırmaktır. İşçi ajanlar problemi farklı açılardan araştırabilir, riskleri bulabilir ve alternatif çözüm önerileri geliştirebilir. Ancak kod değişikliği, test güncellemesi ve son bütünleştirme tek bir ajan tarafından yapılmalıdır. Bu yaklaşım, çoklu ajan mimarisinin keşif gücünden yararlanırken çakışan değişikliklerin maliyetini sınırlı tutar.

Kullanıcı açısından temel kural şudur: çoklu ajan sisteminde yönetici ajan yalnızca görev dağıtan bir ara katman değildir; kapsamı belirleyen, sınırları koruyan, çakışmaları görünür kılan ve nihai kararı birleştiren merkezî denetim noktasıdır. Yönetici–işçi ilişkisinin genel görünümü Şekil 5'te sunulmuştur. İşçi ajanların başarılı olması için görevleri küçük, bağımsız ve ölçülebilir olmalı; çakışma ihtimali bulunan kod değişiklikleri ise doğrudan paralel uygulanmak yerine öneri, karşılaştırma ve tekil uygulama akışıyla yönetilmelidir.




*Şekil 5. Orkestratör-İşçi Deseni*

## Ne Zaman Çoklu Ajan?

Çoklu ajan mimarisinin değer ürettiği görevlerin ortak özelliği, geniş ve paralelleştirilebilir keşiftir. Birden fazla yönde eş zamanlı araştırma yapılması, büyük bir kod tabanının farklı bölgelerinde bağımsız incelemeler yürütülmesi veya alternatif çözüm yollarının karşılaştırmalı olarak denenmesi bu kapsama girer. Bu tür görevlerde alt ajanlar, aynı problemi sırayla çözmeye çalışan işçiler gibi değil, farklı yönlere gönderilen keşif ekipleri gibi çalışır. Her biri kendi bağlam penceresinde ayrı bir hattı araştırır, bulgularını özetler ve ana ajan bu bulguları birleştirerek nihai kararı verir.

Buna karşılık, adımların birbirine sıkı bağımlı olduğu tipik kod yazma işlerinde çoklu ajan kullanımı çoğu zaman beklenen faydayı sağlamaz. Bir dosyadaki değişiklik başka bir dosyadaki değişikliği etkiliyorsa, veri modeli değişmeden servis katmanı güncellenemiyorsa veya testlerin hangi biçimde yazılacağı önceki uygulama kararına bağlıysa, paralel çalışan ajanların üreteceği çıktılar kolayca çakışabilir. Bu durumda asıl maliyet kod yazma işinden çok, ajanların ürettiği parçaları birleştirme işine kayar. Anthropic de çoklu ajan sistemlerinin özellikle eş zamanlı yürütülebilen bağımsız yönler içeren araştırma görevlerinde başarılı olduğunu; çoğu kodlama görevinin ise araştırmaya göre daha az paralelleştirilebilir olduğunu belirtmektedir [12].

Token maliyeti de bu kararın merkezinde yer almalıdır. Anthropic'in deneyimine göre ajanlı sistemler, sıradan sohbet etkileşimlerine kıyasla yaklaşık dört kat; çoklu ajan sistemleri ise yaklaşık on beş kat daha fazla token tüketebilmektedir [12]. Bu fark yalnızca fatura maliyeti anlamına gelmez. Daha fazla token, daha fazla ara çıktı, daha uzun bağlam, daha fazla araç çağrısı ve daha fazla eş güdüm ihtiyacı demektir. Dolayısıyla çoklu ajan mimarisi, ancak görevin değeri bu ek maliyeti karşılıyorsa rasyonel bir tercih hâline gelir.

Bu nedenle çoklu ajan kullanımı, varsayılan çalışma biçimi değil, seçici kullanılan bir ölçekleme aracıdır. Basit bir hata düzeltme, küçük bir refaktör veya tek dosyada yapılacak sınırlı bir değişiklik için birden fazla ajan başlatmak genellikle gereksizdir. Buna karşılık, "bu büyük modülde performans sorunları nerelerden kaynaklanıyor olabilir?", "aynı özelliği üç farklı mimari yaklaşımla nasıl tasarlayabiliriz?" veya "bu kod tabanında güvenlik açısından riskli noktalar hangi katmanlarda yoğunlaşıyor?" gibi keşif ağırlıklı sorularda alt ajanlar değer üretebilir.

Buradan kullanıcı için şu karar kuralı çıkarılabilir: görev "önce geniş keşif, sonra dar uygulama" biçimindeyse keşif aşaması alt ajanlara dağıtılmalı, uygulama kararı ve kod değişikliği tek bir ana ajan üzerinde toplanmalıdır. Görev baştan sona sıralı ilerliyorsa, yani her adım bir önceki adımın sonucuna bağlıysa, tek ajanla ve iyi bağlam disipliniyle çalışmak daha güvenli ve daha ekonomiktir. Çoklu ajan mimarisi bir hızlandırıcıdır; ancak yalnızca yollar birbirinden gerçekten ayrılabiliyorsa hız kazandırır.

# Değerlendirme ve Gözlemlenebilirlik

Harness üzerinde yapılan her değişiklik (yeni bir araç, farklı bir sistem istemi, değiştirilen bir izin politikası) aslında bir hipotezdir ve hipotezler ölçülmeden doğrulanamaz. Değerlendirme (Evals) ve gözlemlenebilirlik, harness mühendisliğini deneme yanılmadan mühendisliğe dönüştüren iki disiplindir.

## Değerlendirme (Evals)

Alanın ortak ölçütü olan SWE-bench ve iyileştirilmiş varyantı SWE-bench Verified önceki bölümlerde tanıtılmıştır ([8], [9]). Ancak bu tür herkese açık kıyaslama setleri, kullanıcının kendi çalışma bağlamı hakkında sınırlı bilgi verir. Modellerin bu kıyaslamalara aşırı uyum sağlaması (benchmark saturation) ve görevlerin kullanıcının gerçek iş yükünü tam olarak yansıtmaması, sonuçların genellenebilirliğini azaltır. Bu nedenle en doğru yaklaşım, her ekibin kendi geliştirme süreçlerini temsil eden değerlendirme (eval) setlerini oluşturmasıdır.

İyi bir değerlendirme setinin büyük olmasına gerek yoktur. Geçmişte tamamlanmış on–yirmi gerçek görev çoğu zaman yeterlidir. Önemli olan, bu görevlerin ekibin günlük çalışmalarını temsil etmesidir. Örneğin sık karşılaşılan bir hata düzeltme, yeni bir API geliştirilmesi, birim testi eklenmesi veya mevcut bir özelliğin yeniden düzenlenmesi gibi işler değerlendirme setine dâhil edilebilir. Böylece ölçülen performans, yapay olarak hazırlanmış örnekler yerine gerçek çalışma koşullarını yansıtır.

Bu değerlendirme seti, harness üzerinde yapılan her anlamlı değişiklikten sonra yeniden çalıştırılmalıdır. Sistem isteminin güncellenmesi, yeni bir araç eklenmesi, farklı bir model kullanılması veya bağlam yönetiminde yapılan değişiklikler gibi her düzenleme aynı görevler üzerinde tekrar değerlendirilmelidir. Böylece değişikliğin gerçekten iyileşme sağlayıp sağlamadığı nesnel olarak görülebilir.

Değerlendirme yalnızca görevin tamamlanıp tamamlanmadığını ölçmemelidir. Başarı oranı, çözüm için gereken tur sayısı, kullanılan token miktarı, tamamlanma süresi ve gerektiğinde insan müdahalesi gibi ölçütler birlikte izlenmelidir. Örneğin başarı oranı değişmezken token maliyetinin yarıya düşmesi veya aynı maliyetle daha az turda sonuca ulaşılması da önemli bir iyileşmedir. Buna karşılık yalnızca başarılı görev sayısına bakmak, verimlilikteki değişimleri gözden kaçırabilir.

Önceki bölümlerde aktarılan METR deneyi bu disiplinin neden gerekli olduğunu göstermektedir. Kullanıcıların “bu sürüm daha hızlı hissettiriyor” veya “ajan artık daha iyi çalışıyor” şeklindeki öznel değerlendirmeleri güvenilir bir performans göstergesi değildir. Aynı görevlerin düzenli olarak yeniden çalıştırılması ve sonuçların sayısal olarak karşılaştırılması, harness üzerinde yapılan değişikliklerin gerçek etkisini ortaya koyan en güvenilir yöntemdir [10].

## İzleme, Telemetri ve Maliyet

Gözlemlenebilirlik tarafında ilk adım, oturum kayıtlarının (transcript) düzenli incelenmesidir. Ajanın hangi araçları hangi sırayla çağırdığı, nerede gereksiz tur harcadığı, hangi araç sonuçlarının bağlamı şişirdiği ancak kayıttan okunabilir. Anthropic’in çoklu ajan sistemi raporunda da üretim güvenilirliğinin anahtarı olarak tam kayıt ve ayrıntılı izleme gösterilmektedir [12]. Kurumsal ölçekte bu izleme standartlaşmaktadır: modern kodlama ajanları OTel [OpenTelemetry (Açık Telemetri)] üzerinden token tüketimi, araç çağrısı sayısı, oturum süresi ve maliyet metriklerini dışa aktarabilmekte; böylece ekipler ajan kullanımını tıpkı diğer üretim sistemleri gibi panolarla izleyebilmektedir. Maliyet görünürlüğü özellikle kritiktir: çoklu ajan desenleri ve uzun oturumlar token tüketimini katlar; hangi görev tipinin ne kadara mal olduğu bilinmeden ne model seçimi ne de otomasyon seviyesi rasyonel biçimde ayarlanabilir. Kısacası harness’ta ölçülmeyen hiçbir şey iyileştirilemez; önce görünürlük sağlanmalı, optimizasyon ondan sonra gelmelidir.



# Kullanıcılar için Harness Yapılandırma Pratikleri

Önceki bölümlerde harness'ın bileşenleri tek tek incelendi. Bu bölümde ise bütün bu kavramlar, bir kodlama ajanı kullanıcısının kendi projesinde izleyebileceği somut bir yapılandırma pratiğine dönüştürülecektir. Buradaki önerilerin ortak ilkesi, harness yapılandırmasının bir kerelik kurulum değil, kod tabanıyla birlikte yaşayan ve sürüm kontrolünde tutulan bir mühendislik varlığı olduğudur.

## AGENTS.md ve Proje Bağlamının Kurulması

Yapılandırmanın ilk adımı, projenin kök dizinine ajanın her oturumda okuyacağı bir bağlam dosyası koymaktır [4]. İyi bir AGENTS.md kısa, açık ve emir kipinde yazılır. Projenin nasıl derleneceği ve test edileceği, dizin yapısının mantığı, kodlama standartları ve ajanın sık yaptığı hatalara karşı uyarılar bu dosyada yer alır. En yaygın hata ise AGENTS.md dosyasını zamanla bir dokümantasyon deposuna dönüştürmektir. Oysa bu dosya her oturumun başında bağlama yüklendiğinden, içindeki her cümle bağlam penceresinden pay alır. Bu nedenle burada "bilinmesi faydalı" bilgiler değil, "yanlış yapılmaması gereken" kurallar bulunmalıdır [3].

AGENTS.md, proje dokümantasyonunun tamamı değildir; daha çok ajanın çalışmaya başlamadan önce okuduğu kısa bir çalışma talimatı niteliğindedir. Nasıl ki bir teknisyen işe başlamadan önce masadaki tüm kullanım kılavuzlarını değil, yalnızca güvenlik talimatlarını ve iş emrini okursa, ajan da öncelikle bu dosyayı okuyarak projeye ilişkin temel kuralları öğrenir. Ayrıntılı tasarım kararları, mimari açıklamalar ve kapsamlı teknik belgeler ise yalnızca gerektiğinde açılmalıdır.

Bu nedenle iyi bir AGENTS.md dosyası, ayrıntıyı kendi içinde barındırmak yerine ilgili dokümanlara yönlendirmeler içerir. Mimari değişikliği yapılacaksa mimari dokümanı, test yazılacaksa test rehberi, güvenlik ile ilgili işlem yapılacaksa güvenlik kuralları okunur. Böylece ajan, ihtiyaç duymadığı bilgileri her oturumda taşımak yerine yalnızca ilgili görev sırasında bağlama alır. Bu yaklaşım "tembel okuma" (lazy-read) olarak adlandırılır ve bağlam bütçesinin verimli kullanılmasını sağlar [3].

Gerçek bir kurumsal Kotlin/Gradle projesinin AGENTS.md dosyasından alınan ve Şekil 6'da sunulan kesit, bu yaklaşımın uygulamadaki görünümünü göstermektedir. Dosya yalnızca temel yönlendirmeleri içerirken ayrıntılar ilgili referans dokümanlara bırakılmıştır.

```
# Repository Guidelines
> Lazy-read policy: Do NOT read every referenced document
> upfront. Only open a document when the current task
> touches that topic.
## Referenced Documents
| Konu                | Doküman              | Ne Zaman Okunmalı                                  |
|---------------------|----------------------|----------------------------------------------------|
| Hexagonal Mimari    | ref/hexagonal.md     | Paket/port/adapter değişince                       |
| Test Yazımı         | ref/testing.md       | Test eklenince/değişince                           |
| Veritabanı          | ref/database.md      | Migration veya şema güncellenirken                 |
| Gradle              | ref/gradle.md        | Modül, bağımlılıklar değiştirilirken               |
| Jenkins CI          | ref/ci.md            | CI/CD hattı ve derleme  değişince                  |
```

*Şekil 6. Kurumsal Bir Projeye Ait AGENTS.md Dosyasından Bir Kesit*

Bu yapı iki önemli avantaj sağlar. Birincisi, her oturumda bağlama giren çekirdek dosya küçük kaldığı için bağlam bütçesi gereksiz bilgilerle tüketilmez. İkincisi, mimari veya test standartları gibi ayrıntılar tek bir yerde tutulduğu için güncelliğini korur; aynı bilginin farklı dosyalarda tekrar edilmesi önlenir.

AGENTS.md yaşayan bir belge olarak görülmelidir. Ajan bir görev sırasında yanlış bir varsayım yaptığında veya ekip aynı hatayla birden fazla kez karşılaştığında, çözümün dosyaya kısa ve net bir kural olarak eklenmesi yerinde olur. Böylece dosya zamanla yalnızca proje kurallarını değil, ekibin ajanla çalışırken edindiği deneyimi de yansıtan ortak bir hafızaya dönüşür.

Monorepo yapılarında ise kök dizindeki genel AGENTS.md dosyasına ek olarak alt dizinlerde modüle özgü AGENTS.md dosyaları kullanılabilir. Bu sayede ödeme sistemiyle ilgili kurallar yalnızca o modülde çalışılırken, mobil uygulamaya ait kurallar ise yalnızca ilgili dizinde görev yapılırken bağlama yüklenir. Kullanıcı açısından temel kural şudur: ortak kurallar kök dizinde, modüle özgü kurallar ise ilgili alt dizinde tutulmalı; her dosya yalnızca gerçekten ihtiyaç duyulan bilgiyi içermelidir.

## Özelleştirilmiş Komutlar ve Yetenekler (Skills)

Tekrarlanan iş akışları, her seferinde yeniden yazılan istemler yerine harness'a kalıcı olarak öğretilmelidir. Aksi hâlde kullanıcı, aynı talimatı farklı oturumlarda yeniden kurmak zorunda kalır. Bu da hem zaman kaybı oluşturur hem de çıktı tutarlılığını zayıflatır. Bu ihtiyacı karşılayan iki temel mekanizma vardır: özelleştirilmiş komutlar (slash commands) ve yetenekler (skills).

Özelleştirilmiş komutlar, sık kullanılan istem şablonlarını kısa çağrılarla erişilebilir hâle getirir. Örneğin kullanıcı /review yazarak bir kod gözden geçirme akışını başlatabilir, /release-notes ile sürüm notu hazırlatabilir ya da /debug ile hata ayıklama sürecini açabilir. Bu yapı, tek bir talimatı tekrar tekrar yazma yükünü ortadan kaldırır. Komutun görevi, "aynı işi her defasında yeniden tarif etmek" yerine, önceden tanımlanmış bir başlangıç kapısı sunmaktır. Kullanıcı açısından kural şudur: iş akışı kısa ise ve tek bir amaca hizmet ediyorsa, onu bir komuta dönüştürmek yeterlidir.

Yetenekler (skills) ise bundan daha kapsamlıdır. Bunlar yalnızca bir başlangıç ifadesi değil, bir iş akışının nasıl yürütüleceğini anlatan, gerektiğinde yüklenen talimat paketleridir [21]. Bir yetenek; adımları, araç kullanımını, ara kontrolleri ve kalite ölçütlerini birlikte içerir. Örneğin "PR [Pull Request (Değişiklik Önerisi)] inceleme" yeteneği, değişiklik listesini okuma, riskli alanları belirleme, test sonuçlarını denetleme ve sonunda yapılandırılmış bir değerlendirme üretme adımlarını içerebilir. Benzer biçimde "bağımlılık yükseltme" yeteneği, güncellenecek paketlerin belirlenmesi, sürüm uyumluluğunun denetlenmesi, değişikliklerin uygulanması ve test paketinin çalıştırılarak sonucun doğrulanması gibi birden fazla adımı kapsayabilir. Kullanıcı açısından kural şudur: iş akışı çok adımlı ise, karar noktaları içeriyorsa ve tekrar kullanılacaksa, onu bir yetenek olarak tanımlamak daha doğrudur.

Bu ayrımın pratik önemi büyüktür. Komut, bir ışık düğmesi gibidir: basılır ve tek bir şey başlar. Yetenek ise bir alet çantası gibidir: yalnızca gerekli olduğunda açılır ve içinde bir işi tamamlamak için gereken parçalar bulunur. Bu nedenle komutlar hızlı ve basit tekrarlar için uygundur; yetenekler ise daha uzun, daha hassas ve yeniden kullanılabilir iş akışları için tercih edilir. Aynı istemin üçüncü kez yazıldığı fark edildiğinde onu bir komuta dönüştürmek yerinde olur. Aynı çok adımlı süreç üçüncü kez tarif edildiğinde ise bir yetenek oluşturmak daha doğru bir yaklaşımdır.

Yeteneklerin etkinliği büyük ölçüde açıklamalarının kalitesine bağlıdır. Harness, hangi yeteneğin yüklenmesi gerektiğine bu kısa açıklamalara bakarak karar verir. Bu nedenle açıklama, yeteneğin yalnızca ne yaptığını değil, hangi durumda kullanılacağını da açıkça belirtmelidir.

Örneğin "Java kodunda performans darboğazlarını inceleme" açıklaması, "kod analizi" gibi genel bir ifadeden daha işlevseldir; çünkü doğru yeteneğin seçilmesini kolaylaştırır. Yetenek içeriği de tek parça, uzun ve dağınık bir talimat bloğu şeklinde değil, adım adım ilerleyen yapılandırılmış bir belge olarak düzenlenmelidir. Böyle bir yapı, hem bakım kolaylığı sağlar hem de modelin görevi yanlış yorumlama ihtimalini azaltır.

Özetle, komutlar tekrar eden kısa çağrıları sadeleştirir; yetenekler ise karmaşık iş akışlarını standartlaştırır. Biri kapıyı açan anahtardır, diğeri o kapıdan girildikten sonra izlenecek yolu gösteren haritadır. Kullanıcı için doğru seçim, işin uzunluğuna ve karmaşıklığına göre yapılmalıdır.

## İzin ve Otomasyon Seviyesinin Ayarlanması

Güvenlik ve izin modeli "Güvenlik ve İzin Modeli" bölümünde kavramsal olarak ele alınmıştı; burada aynı mekanizmalar kullanıcının günlük yapılandırma pratiği açısından değerlendirilir. Varsayılan izin ayarları kullanıcının değil, ajan üreticisinin ne kadar risk göze aldığını yansıtır. İzin politikasının görev türüne göre kademelendirilmesi önerilmektedir. Salt okunur keşif ve analiz görevlerinde geniş otomatik izin verilebilir. Kod değiştiren görevlerde dosya yazma işlemleri serbest bırakılabilir; ancak paket kurulumu, ağ erişimi ve yıkıcı komutlar onaya tabi tutulmalıdır. Üretim sistemlerine erişim sağlayan işlemlerde ise sıkı onay uygulanmalıdır ([15], [20]). Otomasyon seviyesi, doğrulama altyapısının gücüyle birlikte artırılmalıdır: kapsamlı test paketi ve iyi yapılandırılmış bir sanal alan varken ajanı uzun süre gözetimsiz çalıştırmak makuldür; doğrulama zayıfken aynı serbestlik ciddi risk taşır. İstem enjeksiyonu riski taşıyan kaynaklarla (üçüncü taraf MCP sunucuları, web içeriği, dış katkıcıların kodları) çalışırken ölümcül üçlünün üç bileşeninin aynı anda açık olmamasına dikkat edilmelidir [14].

Bu kademeli ve görev temelli izin yaklaşımı, gerçek bir opencode.json yapılandırmasında şöyle somutlaşır: geri döndürülmesi zor komutlar insan onayına ("ask") bağlanırken güvenli komutlar otomatik çalıştırılmaktadır. İlgili yapılandırma örneği Şekil 7'de sunulmuştur.

```
{
  "permission": {
    "bash": {
      "git checkout *": "ask",
      "git commit *": "ask",
      "git push *": "ask",
      "git reset *": "ask",
      "git rebase *": "ask",
      "rm -rf *": "ask"
    }
  }
}
```

*Şekil 7. opencode.json Dosyasından İzin Yapılandırması Örneği*

Her komut için ayrı kural tanımlanabildiğinden (ayrıntılı, yani granular izin), dal geçişi (git checkout) ve toplu silme (rm -rf) gibi yıkıcı işlemler onaya tabi tutulurken sık kullanılan güvenli komutlar akışı kesmeden çalışır.

## Ekip Düzeyinde Harness Standartlaşması

Bireysel yapılandırma alışkanlıkları ekip ölçeğine taşınmadığında, aynı kod tabanında çalışan geliştiriciler birbirinden farklı kalitede ajan çıktıları alır. Bu durum yalnızca kişisel verimlilik farkı üretmez; kod incelemesi, test tekrarı ve davranış tutarlılığı açısından da ekip içinde parçalanmış bir çalışma biçimi doğurur. Yapılandırmasını oturtmuş ekipler harness yapılandırmasını kod gibi yönetir: AGENTS.md dosyaları, özelleştirilmiş komutlar, yetenek tanımları, izin politikaları ve MCP sunucu listeleri depoda versiyonlanır, değişiklikleri kod incelemesinden geçer ve tıpkı CI yapılandırması gibi ekip mülkiyetine dönüşür.

Bu yaklaşımın iki önemli sonucu vardır. Birincisi, yeni ekip üyesi projeye katıldığında yalnızca kodu değil, o kodla nasıl çalışacağını bilen bir ajan kurulumunu da hazır bulur. İkincisi, harness iyileştirmeleri tek bir kişinin kullanım kolaylığıyla sınırlı kalmaz; daha iyi bir komut, daha isabetli bir kural veya daha güvenli bir izin ayarı bütün ekibin çalışma biçimine yansır. Böylece harness, bireysel tercihlerin toplandığı dağınık bir yapı olmaktan çıkar ve ekip standardının parçası hâline gelir.

Anthropic'in 2026 eğilim raporunda, önümüzdeki dönemde ekipler arasındaki verimlilik farkını model erişiminin değil, bu tür harness varlıklarının olgunluğunun belirleyeceği değerlendirilmektedir [17].

# Örnek Ajan Uygulaması İncelemesi: OpenCode

Önceki bölümlerde harness'ın bileşenleri kavramsal olarak ele alınmıştır. Bu bölümde ise bu çerçeve, açık kaynaklı bir kodlama ajanı olan OpenCode üzerinden somutlaştırılacaktır. OpenCode'un örnek ajan uygulaması olarak seçilmesinin başlıca iki nedeni vardır. Birincisi, MIT [Massachusetts Institute of Technology (Massachusetts Teknoloji Enstitüsü)] lisanslı açık kaynak bir proje olması sayesinde harness'ın iç işleyişinin (sistem istemleri, araç tanımları ve ajan döngüsü gibi) doğrudan kaynak kod üzerinden incelenebilmesidir. İkincisi ise model bağımsız tasarımının, bu çalışmada vurgulanan model ile harness arasındaki ayrımı (bkz. "Model ve Ajan Arasındaki Fark") mimari düzeyde açık biçimde göstermesidir ([22], [23]).

## Genel Bakış

OpenCode, SST ekibi (bugünkü adıyla Anomaly) tarafından geliştirilen açık kaynaklı bir kodlama ajanıdır. 2026 yılı itibarıyla en yaygın kullanılan açık kaynaklı terminal tabanlı kodlama ajanlarından biri hâline gelmiştir [23]. Mimari olarak istemci-sunucu ayrımına dayanır. Çekirdek bileşen, ajan döngüsünü ve araç katmanını barındıran bir HTTP [Hypertext Transfer Protocol (Hiper Metin Aktarım Protokolü)] sunucusudur. TUI [Terminal User Interface (Terminal Kullanıcı Arayüzü)], masaüstü uygulaması ve IDE eklentisi ise bu sunucuya bağlanan farklı istemcilerdir [22]. Bu mimari, harness'ın belirli bir kullanıcı arayüzünden bağımsız olarak çalışan bir mühendislik katmanı olduğu yönündeki tezi (bkz. "Harness'ın Doğuşu ve Dönüşümü") somut biçimde göstermektedir.

OpenCode'un bir diğer önemli özelliği sağlayıcı bağımsızlığıdır. Anthropic, OpenAI ve Google modellerinin yanı sıra Ollama üzerinden çalışan yerel modeller dâhil 75'ten fazla sağlayıcıyı desteklemekte ve aynı proje içinde görevin niteliğine göre farklı modeller kullanılabilmektedir [22]. Kod ve bağlam verilerinin üretici sunucularında saklanmaması, yerel modellerle birlikte kullanıldığında tüm ajan döngüsünün kurum içinde yürütülmesine olanak sağlar. Bu özellik, veri gizliliğinin önemli olduğu ortamlarda OpenCode'u uygun bir seçenek hâline getirmektedir [22].

## Harness Anatomisi Açısından OpenCode

"Harness'ın Anatomisi" bölümünde tanıtılan harness katmanları OpenCode üzerinde doğrudan gözlemlenebilmektedir. Ajan döngüsü, varsayılan olarak Build ve Plan olmak üzere iki temel çalışma modu (mode) üzerinden yürütülür. Plan modunda ajan dosya değiştiremez ve terminal komutları kullanıcı onayına tabidir. Bu yaklaşım, "Plan Modu ve İnsan Onayı Kapıları" başlığında açıklanan plan modu ve insan onayı mekanizmalarının ürün düzeyindeki karşılığıdır [22].

Araç katmanında dosya okuma ve yazma, terminal komutları erişimi, kod tabanı araması gibi standart araçların yanında OpenCode'u benzerlerinden ayıran temel özellik LSP entegrasyonudur. Harness, proje için gerekli dil sunucularını otomatik olarak başlatır ve her düzenlemeden sonra derleyici tanılamalarını modele geri besler. Böylece ajan yaptığı değişikliğin tip hatası oluşturduğunu bir sonraki adımda görebilir ve buna göre kodunu düzeltebilir ([19], [22]). Bu yapı, "Test, Lint ve Mimari Kural Tabanlı Doğrulama" başlığında vurgulanan hızlı ve sürekli geri bildirim ilkesinin araç katmanındaki somut uygulamasıdır.

Doğrulama katmanında ise OpenCode, her anlamlı değişiklikten önce Git tabanlı bir anlık görüntü oluşturur ve yapılan değişikliklerin /undo ve /redo komutlarıyla geri alınmasına olanak tanır. Bu yaklaşım, her işlem için önceden izin istemekten ziyade güvenli geri alınabilirliği esas alan bir güvenlik yaklaşımını yansıtmaktadır [22].

## Yapılandırma Pratikleri

"Kullanıcılar için Harness Yapılandırma Pratikleri" bölümünde önerilen yapılandırma pratiklerinin tamamı OpenCode'da doğrudan karşılık bulmaktadır. Kalıcı bağlam için depo köküne yerleştirilen AGENTS.md dosyası varsayılan olarak desteklenmektedir (bkz. "Kalıcı Bağlam: AGENTS.md ve Hafıza") [4]. Proje düzeyindeki opencode.json dosyası ise sağlayıcı ve model seçiminden izin politikalarına kadar harness'ın davranışını tek bir yapılandırma dosyasında toplar ve bu yapılandırmanın sürüm kontrolü altında yönetilmesini sağlar [22]. Özelleştirilmiş komutlar .opencode/commands dizinindeki Markdown dosyalarıyla tanımlanır. Dosya adı doğrudan komut adına dönüşür ve şablon içinde adlandırılmış argümanlar kullanılabilir. Bu yapı, "Özelleştirilmiş Komutlar ve Yetenekler (Skills)" başlığında önerilen "aynı istem üçüncü kez kullanıldığında onu bir komuta dönüştür" ilkesinin uygulama mekanizmasını oluşturmaktadır. Benzer şekilde kullanıcılar; sistem istemi, model ve izin kümesini tanımlayarak yeni ajanlar oluşturabilir. Bu ajanlar birincil ajan veya alt ajan (subagent) olarak yapılandırılabilir. Böylece "Alt Ajanlar (Subagents)" başlığında açıklanan alt ajan deseni kullanıcı tarafından doğrudan uygulanabilir hâle gelmektedir [22]. İzin modeli yalnızca ajan düzeyinde değil, terminal komutları düzeyinde de yapılandırılabilmektedir. Glob desenleri kullanılarak belirli komutlar için ayrı izin kuralları tanımlanabilir. Örneğin tüm komutlar varsayılan olarak kullanıcı onayına tabi tutulurken, git status gibi güvenli komutlara otomatik izin verilebilir. Bu esneklik, "İzin Modelleri ve Otomasyon Seviyeleri" ile "İzin ve Otomasyon Seviyesinin Ayarlanması" başlıklarında önerilen görev temelli ve kademeli izin politikasının doğrudan uygulanmasına olanak sağlamaktadır [22].

Son olarak MCP desteği, "Araçların Harness İçindeki Rolü" başlığında açıklanan protokol aracılığıyla harici araçların harness'a eklenmesini mümkün kılmaktadır [6].

## Diğer Ajanlarla Karşılaştırma

OpenCode'u ticari muadilleriyle karşılaştırmak, harness tasarımında benimsenen farklı öncelik tercihlerini somut biçimde ortaya koyar. Claude Code tek sağlayıcıya (Anthropic) bağlıdır ve gücünü, model ile harness'ın aynı elden ve birbirine göre optimize edilmesinden alır. Bu, donanımı ve yazılımı aynı çatı altında tasarlayan iPhone yaklaşımına benzer: her parça diğerine göre ayarlandığından deneyim akıcı, tutarlı ve öngörülebilir olur. Hız, olgunluk ve uzun otonom görevlerdeki kararlılık bu bütünleşik tasarımın sonuçlarıdır [20]. Codex CLI ve Gemini CLI de benzer biçimde kendi üreticilerinin modelleri etrafında şekillenir. OpenCode ise karşı uçtaki tasarım felsefesini temsil eder. Sağlayıcı bağımsızlığı, açık kaynak denetlenebilirlik, çoklu arayüz ve LSP geri beslemesi gibi harness düzeyi yeteneklerle ayrışır ([22], [23]). Bu tercihlerin bedelleri de vardır: bağımsız değerlendirmeler, aynı model kullanıldığında OpenCode'un görevleri Claude Code'a kıyasla belirgin biçimde daha yavaş tamamladığını göstermektedir [24]; bu farkın önemli bölümü istemci-sunucu mimarisinden, LSP geri besleme döngüsünden ve OpenCode'un daha fazla test yazmak gibi daha kapsamlı davranmasından kaynaklanır. Buna karşılık hız tek başına kalite anlamına gelmez: aynı modelle yürütülen kimi bağımsız kıyaslamalarda OpenCode, görev doğruluğu ve kapsamlılık açısından Claude Code ile en az aynı düzeyde, hatta bazı ölçümlerde daha yüksek başarı elde etmektedir ([24], [25]). Ayrıca hızlı sürüm temposunun zaman zaman kararlılık sorunları getirebildiği bildirilmektedir. Kullanıcı açısından pratik seçim kuralı şöyle özetlenebilir: model seçme özgürlüğü, maliyet kontrolü veya veri gizliliği belirleyici ise OpenCode güçlü bir varsayılandır; tek sağlayıcının bütünleşik deneyimi ve hızı öncelikli ise ticari araçlar öne geçer. Bununla birlikte maliyet, harness'lar arasındaki seçimi etkileyen ayrı bir kısıttır: ticari araçlar sabit abonelik modeliyle sunulmakta, üçüncü taraf harness'larda ise model erişimi kullanıma bağlı (token bazlı) olarak ücretlendirilmektedir. Ajanın her adımı ek token tükettiğinden yoğun kullanımda toplam maliyet, sabit abonelik ücretinin üzerine çıkabilir. Bu nedenle, özellikle kurumsal ölçekte yaygın kullanımda, yerel modellerle çalışabilen OpenCode gibi açık kaynak çözümler maliyet etkinliği açısından öne çıkar. Rapor boyunca savunulan çerçeveyle söylenirse bu, "hangisi daha iyi" sorusu değil, hangi harness önceliklerinin hangi kullanım bağlamıyla örtüştüğü sorusudur.



# Sonuç ve Öneriler

Bu çalışmada, kodlama ajanlarının çalışma biçimini belirleyen harness katmanı; temel bileşenleri, mimari yapısı ve yazılım geliştirme süreçlerine etkisi açısından incelenmiştir. İnceleme sonucunda iki temel çıkarıma ulaşılmıştır.

Çalışmanın temel tezi, kodlama ajanlarında gözlenen performansın tek başına modelin değil, model ile onu saran harness'ın ortak ürünü olduğudur. Sistem istemi, araç katmanı, bağlam yönetimi, doğrulama döngüleri, izin modeli ve çoklu ajan koordinasyonundan oluşan bu katman, aynı modelden bambaşka sonuçlar çıkarabilmektedir; SWE-bench Verified'ın iskele (scaffold) farklarına ilişkin bulguları ve Anthropic'in çoklu ajan deneyimleri bunu nicel olarak doğrulamaktadır. Buna göre ajanından memnun olmayan bir kullanıcının bakması gereken ilk yer model seçimi değil, kendi harness yapılandırmasıdır. Bağlam dosyaları, araç seti, izin politikası ve doğrulama altyapısının tamamı kullanıcının kontrolündeki ayar noktalarıdır ve bu çalışma boyunca gösterildiği üzere, her biri çıktı kalitesi üzerinde ölçülebilir etkiye sahiptir. "Örnek Ajan Uygulaması İncelemesi: OpenCode" bölümündeki inceleme de bu tezi somut bir ürün üzerinde doğrulamaktadır: çalışmada anlatılan katmanların her biri (ajan döngüsü, araç katmanı, izin modeli, kalıcı bağlam) açık kaynaklı bir harness'ta birebir gözlemlenebilir ve kullanıcı tarafından yapılandırılabilir durumdadır.

İkinci temel çıkarım, harness mühendisliğinin ölçüm olmadan yapılamayacağıdır. METR deneyinin ortaya koyduğu algı–gerçeklik farkı, yani geliştiricilerin kendilerini %20 daha hızlı sanırken ölçümde %19 daha yavaş çıkması, öznel değerlendirmenin bu alanda ne kadar yanıltıcı olduğunun en çarpıcı kanıtıdır. Harness üzerinde yapılan her değişiklik küçük ve gerçekçi bir değerlendirme setiyle sınanmalı; otomasyon seviyesi ancak doğrulama altyapısı (testler, lint, sanal alan) güçlendikçe artırılmalıdır. Güçlü geri bildirim döngüleri kurulmadan verilen geniş özerklik ise hata üretimini hızlandırmaktan öteye geçmemektedir.

Kurumların, harness yapılandırmasını bireysel alışkanlık olmaktan çıkarıp versiyonlanan bir mühendislik varlığına dönüştürmesi önerilmektedir. AGENTS.md dosyaları, özelleştirilmiş komutlar, yetenek tanımları, izin politikaları ve onaylı MCP sunucu listesi depoda tutulmalı, değişiklikleri kod incelemesinden geçmeli ve ekip mülkiyetinde olmalıdır. Güvenlik tarafında istem enjeksiyonu, teorik bir merak değil OWASP'ın bir numaralı LLM riski olarak sınıflandırdığı güncel bir tehdittir; özel veriye erişim, güvenilmeyen içerik ve dışa iletişimden oluşan ölümcül üçlünün aynı anda açık olmaması ilkesi, kurumsal izin politikalarının temeline yerleştirilmelidir. Üçüncü taraf MCP sunucuları, tıpkı yazılım bağımlılıkları gibi tedarik zinciri denetiminden geçirilmelidir.

Bireysel kullanıcıların ise harness okuryazarlığını temel bir mesleki beceri olarak benimsemesi önerilmektedir. Bu okuryazarlığın somut adımları şunlardır: Her projeye kısa ve emir kipinde bir AGENTS.md ile başlamak; ajanın yanlış varsayımlarını bu dosyaya tek satırlık düzeltmeler olarak işlemek; tekrarlanan istemleri komutlara, tekrarlanan iş akışlarını yeteneklere dönüştürmek; oturum kayıtlarını düzenli okuyarak ajanın nerede tur ve token israf ettiğini görmek; ve bağlam penceresini sınırlı bir bağlam bütçesi olarak yönetmeyi öğrenmek. Bu beceriler, belirli bir üründen bağımsızdır: MCP, ACP ve AGENTS.md gibi standartlar sayesinde bir harness'ta kurulan yapılandırma ve kazanılan sezgi, büyük ölçüde diğerlerine taşınabilmektedir.

Son olarak, bu alanın olağanüstü hızla değiştiği unutulmamalıdır. Bugün en iyi pratik sayılan bir desen altı ay sonra gereksizleşebilir; modellerin yetkinliği arttıkça harness'ların bazı katmanları, örneğin ayrıntılı yönlendirme, önemi görece azalırken doğrulama, güvenlik ve gözlemlenebilirlik gibi katmanlar daha da önem kazanmaktadır. Bu değişkenlik, harness mühendisliğini öğrenmeyi anlamsız kılmaz; tam tersine, bileşenleri ve ilkeleri kavramış bir kullanıcı, hangi ürün ya da model gelirse gelsin yeni duruma hızla uyum sağlayabilir. Harness mühendisliği doğru uygulandığında, kodlama ajanlarını kestirilemez birer kara kutudan, davranışı anlaşılabilir, ölçülebilir ve yönetilebilir mühendislik sistemlerine dönüştürmektedir; bu çalışmanın amacı da kullanıcılara bu dönüşümün haritasını sunmaktır.

# Kaynakça


[1] S. Yao, J. Zhao, D. Yu vd., **"ReAct: Synergizing Reasoning and Acting in Language Models"**, The Eleventh International Conference on Learning Representations (ICLR), 2023. Erişim Adresi: https://arxiv.org/abs/2210.03629, Erişim Zamanı: 22 Eylül 2026.

[2] Anthropic, **"Building Effective Agents"**, 2024. Erişim Adresi: https://www.anthropic.com/research/building-effective-agents, Erişim Zamanı: 22 Eylül 2026.

[3] Anthropic, **"Effective Context Engineering for AI Agents"**, 2025. Erişim Adresi: https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents, Erişim Zamanı: 22 Eylül 2026.

[4] OpenAI, Google, Anthropic vd., **"AGENTS.md: A Simple, Open Format for Guiding Coding Agents"**, 2025. Erişim Adresi: https://agents.md/, Erişim Zamanı: 22 Eylül 2026.

[5] Anthropic, **"Writing Effective Tools for Agents — with Agents"**, 2025. Erişim Adresi: https://www.anthropic.com/engineering/writing-tools-for-agents, Erişim Zamanı: 22 Eylül 2026.

[6] Anthropic, **"Introducing the Model Context Protocol"**, 2024. Erişim Adresi: https://modelcontextprotocol.io/, Erişim Zamanı: 22 Eylül 2026.

[7] Zed Industries, **"Agent Client Protocol"**, 2025. Erişim Adresi: https://agentclientprotocol.com/, Erişim Zamanı: 22 Eylül 2026.

[8] C. E. Jimenez vd., **"SWE-bench: Can Language Models Resolve Real-World GitHub Issues?"**, The Twelfth International Conference on Learning Representations (ICLR), 2024. Erişim Adresi: https://www.swebench.com/, Erişim Zamanı: 22 Eylül 2026.

[9] OpenAI, **"Introducing SWE-bench Verified"**, 2024. Erişim Adresi: https://openai.com/index/introducing-swe-bench-verified/, Erişim Zamanı: 22 Eylül 2026.

[10] J. Becker, N. Rush, E. Barnes ve D. Rein, **"Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity"**, METR, 2025. Erişim Adresi: https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/, Erişim Zamanı: 22 Eylül 2026.

[11] Q. Wu, G. Bansal, J. Zhang vd., **"AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation"**, Microsoft Research, 2023. Erişim Adresi: https://arxiv.org/abs/2308.08155, Erişim Zamanı: 22 Eylül 2026.

[12] Anthropic, **"How We Built Our Multi-Agent Research System"**, 2025. Erişim Adresi: https://www.anthropic.com/engineering/built-multi-agent-research-system, Erişim Zamanı: 22 Eylül 2026.

[13] OWASP, **"OWASP Top 10 for LLM Applications 2025"**, 2025. Erişim Adresi: https://genai.owasp.org/, Erişim Zamanı: 22 Eylül 2026.

[14] S. Willison, **"The Lethal Trifecta for AI Agents"**, 2025. Erişim Adresi: https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/, Erişim Zamanı: 22 Eylül 2026.

[15] OpenAI, **"A Practical Guide to Building Agents"**, 2025. Erişim Adresi: https://openai.com/business/guides-and-resources/a-practical-guide-to-building-ai-agents/, Erişim Zamanı: 22 Eylül 2026.

[16] Google, **"Announcing the Agent2Agent Protocol (A2A): A New Era of Agent Interoperability"**, 2025. Erişim Adresi: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/, Erişim Zamanı: 22 Eylül 2026.

[17] Anthropic, **"2026 Agentic Coding Trends Report"**, 2026. Erişim Adresi: https://resources.anthropic.com/2026-agentic-coding-trends-report, Erişim Zamanı: 22 Eylül 2026.

[18] J. Li vd., **"Agent Harness Engineering: A Survey"**, OpenReview (TMLR başvurusu), 2026. Erişim Adresi: https://openreview.net/pdf?id=eONq7FdiHa, Erişim Zamanı: 22 Eylül 2026.

[19] Microsoft, **"Language Server Protocol"**, 2025. Erişim Adresi: https://microsoft.github.io/language-server-protocol/, Erişim Zamanı: 22 Eylül 2026.

[20] Anthropic, **"Claude Code: Best Practices for Agentic Coding"**, 2025. Erişim Adresi: https://www.anthropic.com/engineering/claude-code-best-practices, Erişim Zamanı: 22 Eylül 2026.

[21] Anthropic, **"Equipping Agents for the Real World with Agent Skills"**, 2025. Erişim Adresi: https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, Erişim Zamanı: 22 Eylül 2026.

[22] Anomaly, **"OpenCode Documentation"**, 2026. Erişim Adresi: https://opencode.ai/docs/, Erişim Zamanı: 22 Eylül 2026.

[23] Anomaly, **"OpenCode: The Open Source AI Coding Agent (GitHub deposu)"**, 2026. Erişim Adresi: https://github.com/anomalyco/opencode, Erişim Zamanı: 22 Eylül 2026.

[24] AIMultiple, **"Agentic Coding CLI Tools: Benchmark and Comparison"**, 2026. Erişim Adresi: https://aimultiple.com/agentic-cli, Erişim Zamanı: 22 Eylül 2026.

[25] Sentry, **"Warden: Coding Agent Benchmarking Analysis"**, 2026. Erişim Adresi: https://warden.sentry.dev/benchmarking, Erişim Zamanı: 22 Eylül 2026.

# Kısaltmalar

| Kısaltmalar | Açıklama |
|---|---|
| **TÜBİTAK** | Türkiye Bilimsel ve Teknolojik Araştırma Kurumu |
| **BİLGEM** | Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi |
| **YTE** | Yazılım Teknolojileri Araştırma Enstitüsü |
| **A2A** | Agent2Agent (Ajanlar Arası) |
| **ACP** | Agent Client Protocol (Ajan İstemci Protokolü) |
| **API** | Application Programming Interface (Uygulama Programlama Arayüzü) |
| **CI/CD** | Continuous Integration / Continuous Deployment (Sürekli Bütünleştirme / Sürekli Dağıtım) |
| **CLI** | Command Line Interface (Komut Satırı Arayüzü) |
| **ETCLOVG** | Execution, Tooling, Context, Lifecycle, Observability, Verification, Governance (Yürütme Ortamı, Araç Arayüzü, Bağlam Yönetimi, Yaşam Döngüsü ve Orkestrasyon, Gözlemlenebilirlik, Doğrulama, Yönetişim) |
| **HTTP** | Hypertext Transfer Protocol (Hiper Metin Aktarım Protokolü) |
| **IDE** | Integrated Development Environment (Bütünleşik Geliştirme Ortamı) |
| **LLM** | Large Language Model (Büyük Dil Modeli) |
| **LSP** | Language Server Protocol (Dil Sunucusu Protokolü) |
| **MCP** | Model Context Protocol (Model Bağlam Protokolü) |
| **METR** | Model Evaluation and Threat Research (Model Değerlendirme ve Tehdit Araştırması) |
| **MIT** | Massachusetts Institute of Technology (Massachusetts Teknoloji Enstitüsü) |
| **OTel** | OpenTelemetry (Açık Telemetri) |
| **OWASP** | Open Worldwide Application Security Project (Açık Dünya Çapında Uygulama Güvenliği Projesi) |
| **PR** | Pull Request (Değişiklik Önerisi) |
| **RAG** | Retrieval-Augmented Generation (Erişim Destekli Üretim) |
| **ReAct** | Reasoning and Acting (Akıl Yürütme ve Eylem) |
| **SWE-bench** | Software Engineering Benchmark (Yazılım Mühendisliği Kıyaslama Ölçütü) |
| **TDD** | Test-Driven Development (Test Odaklı Geliştirme) |
| **TUI** | Terminal User Interface (Terminal Kullanıcı Arayüzü) |
| **VM** | Virtual Machine (Sanal Makine) |

# Şekiller ve Tablolar


| | |
|---|---|
| 10 | Şekil 1. Harness'ın Anatomisi |
| 11 | Şekil 2. Ajan Döngüsü |
| 14 | Şekil 3. Katmanların Bütüncül Görünümü: ETCLOVG Taksonomisi |
| 23 | Şekil 4. Ölümcül Üçlü |
| 25 | Şekil 5. Orkestratör-İşçi Deseni |
| 28 | Şekil 6. Kurumsal Bir Projeye Ait AGENTS.md Dosyasından Bir Kesit |
| 30 | Şekil 7.  opencode.json Dosyasından İzin Yapılandırması Örneği |







İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgi@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)