

Professional English-2

Module 2

Bu dersin ana bileşenleri:

- **Grammer odağı:** 3. tekil şahıs ve akademik üslupta kişi zamirleri (I/you yerine third person)
- **Telaffuz odağı:** Akademik ve teknik kelimeleri doğru söyleme
- **Mesleki metinler:** Academic integrity, essay structure, paragraph structure
- **Dinleme/okuma:** Kısa ders videoları ve okuma parçaları
- **Uygulama:** Tez cümlesi yazma, konu cümlesi yazma, paragraf geliştirme

Bölüm A

A1) Academic Integrity (Akademik Dürüstlük) nedir?

Academic integrity, üniversitede öğrenme ve değerlendirme süreçlerinde **dürüst**, **sorumlu** ve **adil** davranma ilkesidir.

Akademik dürüstlük şunları kapsar

- Ödevi, raporu, projeyi kendin üretmek
- Başkasının cümlesini ya da fikrini kullanıyorsan kaynak göstermek
- Sınav ve ödev yönergelerine uymak
- Başkalarının kopya çekmesine yardım etmemek

BÖLÜM B: Bilimsel Makale (Essay) Nedir?

Essay, tek bir konu etrafında odaklanan ve o konuyu paragraf paragraf geliştiren akademik metindir.

Makalenin Temel Parçaları

- **Title:** Konuyu açıklar.
- **Introduction:** Konuya giriş yapar ve sonunda **thesis statement** vardır.
- **Body paragraphs:** Tezi destekleyen paragraflar (genelde 2–4 paragraf; 3 çok yaygın).
- **Conclusion:** Metni toparlar, özetler veya tezi farklı biçimde yeniden ifade eder.

Yaygın essay türleri

- Illustration: örneklerle gösterme
- Argument: gerekçelerle savunma
- Classification: kategorilere ayırma
- Compare/Contrast: benzerlik-farklılık
- Cause/Effect: neden-sonuç

BÖLÜM C: Introduction Paragraph (Giriş Paragrafı)

C1) Dikkat Çekici Giriş (HOOK) nedir?

Hook, okuyucunun ilgisini çeken ilk cümledir. Hook türleri:

1. Şaşırtıcı bilgi
2. Kısa anekdot
3. İlginç soru (çok sık kullanılmaz)
4. Alıntı (çok bilinen alıntılardan kaçın)
5. İstatistik (çok teknik ve sıkıcı olmamalı)

Software Engineering örnek hook (örnek cümleler):

- “A single software bug can cause millions of dollars in losses.”
- “Many users stop using an app after one bad experience.”

C2) Funnel yöntemi (genelden özele)

Giriş paragrafı genelden başlar, konuya yaklaşır ve en sonda thesis statement ile biter.

Tipik uzunluk: 3–5 cümle.

BÖLÜM D: Thesis Statement (Tez Cümlesi)

D1) Thesis statement’in iki bileşeni

İyi bir thesis statement:

- Topic (konu)
- Controlling idea (konu hakkında savunulan iddia, yön)

D2) Kaçınılması gerekenler

- “This essay will talk about...” gibi duyuru cümleleri
- Tartışılmayan gerçekler
- Herkesin zaten kabul ettiği bariz fikirler
- Beeirsiz zamirler (it, this, they) ile muğlaklık
- “important” gibi aşırı genel controlling idea

D3) iyi thesis örnekleri

- “Code reviews should be mandatory in software teams because they reduce defects, improve knowledge sharing, and increase maintainability.”
- “Microservices are not always the best architecture choice; for small teams they often increase operational complexity more than they improve scalability.”

BÖLÜM E: Geliştirme Paragrafları (Body Paragraf)

E1) Body paragraph yapısı

1. Topic sentence
2. Supporting ideas (2–4 tane, çoğunlukla 3)
3. Her supporting idea için details/examples
4. Transition/closing sentence (sonraki paragrafa bağlar)

E2) Topic sentence nasıl olmalı?

Topic sentence de iki parçalı olmalı:

- topic
- controlling idea

Ayrıca:

- Thesis statement ile bağlantılı anahtar kelimeleri içerir.
- Akademik yazmada gereksiz zamir kullanımından kaçınır.

E3) Akıcılık (Cherence) nedir?

Coherence, cümle ve paragraflar arası akıcılıktır. Şunlarla güçlenir:

- Transition words (first, moreover, however, therefore, in addition)
- Keyword repetition veya synonym kullanımı
- Uygun cümle bağlama (çok kısa kopuk cümleleri birleştirme)

BÖLÜM F: Paragraf Temelleri

F1) Bütünlük (Unity)

Paragraftaki tüm cümleler topic sentence ile ilgili olmalıdır. Alakasız cümle unity bozar.

F2) Bağlaşıklık (Cohesion)

Cümleler birbirine bağlı akmalıdır:

- Geçiş ifadeleri
- Anahtar kelime tekrarları

- Eş anlamlı kelimeler
- Mantıklı referanslar (zamir kullanımı kontrollü)

Agile Development

Agile development improves software delivery by enabling rapid feedback, flexible planning, and continuous improvement. First, short iterations help teams validate requirements early and reduce the risk of building the wrong features. For example, a team can ship a small increment every two weeks and adjust the backlog based on user feedback and stakeholder priorities. Moreover, agile ceremonies such as stand-ups and sprint reviews increase alignment by making progress and blockers visible. When blockers surface quickly, teams can re-plan tasks, coordinate dependencies, and protect the sprint goal. As a result, agile development supports predictable progress while still adapting to changing requirements.

Clean Code

Clean code increases maintainability by making software easier to read, test, and change over time. First, meaningful naming and small, focused functions reduce cognitive load for developers who join the project later. For instance, a function named `calculateInvoiceTotal()` communicates intent better than a vague name like `processData()`, and a short function is easier to review and debug. In addition, consistent formatting and clear separation of concerns prevent “spaghetti code” that spreads logic across unrelated files. When code structure is consistent, teams can locate defects faster and implement changes with fewer side effects. Therefore, clean code is not only a style preference; it is a practical strategy for long-term reliability and sustainable development.

1. Topic sentence
2. Supporting Idea
3. Detail / Example
4. Transition / Closing sentence

Agile Development

Agile development improves software delivery by enabling rapid feedback, flexible planning, and continuous improvement. First, short iterations help teams validate requirements early and reduce the risk of building the wrong features. For example, a team can ship a small increment every two weeks and adjust the backlog based on user feedback and stakeholder priorities. Moreover, agile ceremonies such as stand-ups and sprint reviews increase alignment by making progress and blockers visible. When blockers surface quickly, teams can re-plan tasks, coordinate dependencies, and protect the sprint goal. As a result, agile development supports predictable progress while still adapting to changing requirements.