# The Pragmatic Programmer Özet

## Bölüm 1- Pragmatik Bir Felsefe

1. Bu senin hayatın
   * Bu senin hayatın, sahibi sensin ve alacağın her aksiyon senin sorumluluğunda olacak.
   * Seçim yapma özgürlüğüne sahipsin.
2. Kedi Kaynak Kodumu Yedi
   * Sorumluluk almak
     + Saçma bahaneler yerine seçenekler sunun.
3. Yazılım Entropisi
   * "Kırık pencereleri" (kötü tasarımlar, yanlış kararlar veya zayıf kod) onarmadan bırakmayın.
4. Taş Çorbası ve Haşlanmış Kurbağa
   * Değişim için katalizör olun.
   * Büyük resmi hatırla (Sadece kişisel olarak ne yaptığınızı değil, etrafınızda olup bitenleri sürekli gözden geçirin).
5. Yeterince İyi Yazılım
   * Kullanıcılarını pazarlıklara dahil et
   * Ürettiğiniz sistemin kapsamı ve kalitesi, o sistemin gereksinimlerinin bir parçası olarak belirtilmelidir.
   * Kaliteyi gereksinim sorunu haline getirin
   * Ne zaman duracağınızı bilin (iyi bir durumda olan programı aşırı süslemeye çalışarak bozmaya gerek yok.).
6. Bilgi Portföyünüz
   * Bilgi Portföyünüz
     + Bilgi birikimine yatırım her zaman en iyi faiz öder.
     + Ciddi yatırımcılar, bir alışkanlık olarak düzenli olarak yatırım yapar.
     + Çeşitlendirme, uzun vadeli başarının anahtarıdır.
     + Akıllı yatırımcılar yüksek-riskli ve az-riskli yatırımlar arasında denge gözetirler.
     + Yatırımcılar maksimum kazanç için düşük alıp yüksek satmaya çalışırlar.
     + Portföyler düzenli şekilde gözden geçirilip tekrar düzenlenmelidir.
   * Portföyünüzü Oluşturmak
     + Düzenli olarak yatırım yapın (Finansal yatırım gibi bilgi portföyüne de düzenli yatırım yapmalısın.)
     + Çeşitlendirin. (Ne kadar farklı şey bilirsen, o kadar değerli olursun.)
     + Riski yönetin
     + Düşükten alın yüksekten satın. ( Yeni ortaya çıkan teknolojileri öğrenip kullanmak çok zor bir iştir ama bir o kadar da getirisi olan bir şeydir.).
     + Gözden geçirin ve tekrar düzenleyin. (Geçenlerde kullandığın yeni teknoloji şimdi çok eski olabilir.
   * Bilgi portföyünüze düzenli olarak yatırım yapın.
   * Hedefler
     + Her yıl en az bir yeni dil öğrenin.
     + Her çeyrekte bir teknik kitap okuyun.
     + Teknik olmayan kitapları da okuyun.
     + Ders alın.
     + Yerel kullanıcı gruplarına katılın.
     + Farklı ortamlarla denemeler yapın.
     + Güncel kalın.
     + Aktif olun.
   * Eleştirel Düşünme
     + Okuduklarınızı ve duyduklarınızı eleştirel olarak analiz edin.
7. İletişim kurmak
   * Ne söylemek istediğinizi bilin.
     + Şimdi ne demek istiyorsun. Ne söylemek istediğini planla. Bir taslak yazın.
     + Kendi kendine sor, “Bunlar söylemek istediğim şeyleri karşıya aktarıyor mu?
     + Aktarmıyorsa sadeleştir.
   * Kitlenizi tanıyın
     + Ne istiyorlar?
     + İlgileri nedir?
     + Ne kadar sofistikeler?
     + Ne kadar detay istiyorlar?
     + Bilgiye kimin sahip olmasını İstiyorsunuz?
     + Onları dinlemeye nasıl motive edebilirsiniz?
   * Anınızı seçin: Hedef kitlenizin bilgilerinizi ne zaman duyması gerektiğini anlayın.
   * Bir stil seçin: (Yalnızca gerçekler, geniş ciltli raporlar, basit bir not.)
   * İyi görünmesini sağlayın: Önemli fikirlerinize güzel görünen bir araç ekleyin ve hedef kitlenizin ilgisini çekin.
   * Kitlenizi dahil edin: Geri bildirimlerini alın ve akıllarını karıştırın.
   * Dinleyici olun: Soru sorarak insanları konuşmaya teşvik edin.
   * İnsanlara geri dönün: Daha sonra insanları bilgilendirin.
   * Ne söylediğin kadar nasıl söylediğin de önemlidir.

## Bölüm 2-Pragmatik Bir Yaklaşım

1. İyi Tasarımın Özü
   * İyi tasarım kötü tasarımdan daha kolay değiştirilir.
   * Bir şey, onu kullanan insanlara uyum sağlıyorsa iyi tasarlanmış demektir.
   * Easier to change.(Değiştirmek daha kolay)
   * ETC bir kural değil, değerdir.
   * Kendinize “şimdi yaptığım şey genel sistemi değiştirmeyi kolaylaştırdı mı yoksa zorlaştırdı mı?” diye sorun.
   * Yazdıklarınızı değiştirilebilir hale getirmeye çalışın.
   * Kodu ayrıştırılmış ve uyumlu tutmayı her zaman aklınızda tutun.
2. Kopyalamanın Kötülükleri
   * Kendinizi Tekrar Etmeyin
     + Dayatılan çoğaltma: Geliştiricilerin çoğaltmadan başka seçeneği olmadığını düşündüğü zaman ortaya çıkar.
     + Yanlışlıkla yapılan çoğaltma: Geliştiriciler çoğaltma yaptığının farkında olmaz.
     + Sabırsız çoğaltma: Geliştiriciler daha kolay göründüğü için tembelleşir ve kopyalar.
     + Geliştiriciler arası çoğaltma: Bir ekipteki (veya farklı ekiplerdeki) birden çok kişi bir bilgi parçasını çoğaltır.
   * Yeniden kullanımı kolaylaştırın.
   * Kötü kod çok sayıda yorum gerektirir.
   * Kısa yollar uzun geciktirmeler yaratır.
   * İnsanların kendileri yazmalarındansa, var olan şeyleri tekrar bulup kullanmalarını teşvik edecekleri bir ortam kur.
3. Dikeylik (ortogonallik)
   * Birindeki değişiklikler diğerlerini etkilemiyorsa iki veya daha fazla şey dikeydir.
   * İlgisiz şeyler arasındaki etkileri ortadan kaldırın
   * Biz kendi kendine yeten, bağımsız, tek ve iyi-tanımlanmış amacı olan bileşenler tasarlamak istiyoruz.
   * Faydaları:
     + Verimlilik Kazanın
       - Değişiklikler sınırlandırılır.
       - Yeniden kullanımı teşvik eder.
       - Dikey bileşenleri birleştirdiğin zaman verimde çok muazzam bir kazanç elde edersin. Birbirine dikey olan, M farklı şeyi yapan bir bileşen ile N farklı şeyi yapan bir bileşeni birleştirdiğinde M x N şey ortaya çıkar.
     + Azaltılmış Risk
       - Hastalıklı bölümler veya kod izole edilir.
       - Dikey bir sistem daha iyi test edilecektir.
       - Bir ürüne veya platforma bağlı değil
   * İşyerinizde uygulayabileceğiz dikey ilkeler.
     + Proje Ekipleri: İşlevsellik bölünmüştür
     + Tasarım: Komple bir projeyi bileşenleri aracılığıyla tasarlamak daha kolaydır.
     + Araç Setleri ve Kütüphaneler: Dikeyliği korumak için akıllıca seçimler yapın.
     + Kodlama: Kod eklerken dikeyliği korumak için şunları yapın:
       - Kodunuzu ayrı tutun.
       - Küresel verilerden(Global Değişken) kaçının.
       - Benzer fonksiyonlardan kaçının.
     + Test etme: Dikey sistemlerin test edilmesi daha kolaydır.
4. Tersine Çevrilebilme
   * Değişikliklere hazırlıklı olun.
   * Nihai karar yoktur.
5. İzli Mermiler
   * Yeni projelerde, kullanıcılarınızın gereksinimleri belirsiz olabilir. Yeni algoritmaların, tekniklerin, dillerin veya bilinmeyen kitaplıkların kullanımı gelecek. Ve çevre, siz işiniz bitmeden zamanla değişecektir. Bizi bir gereksinimden son sistemin bir yönüne hızlı, görünür ve tekrarlanabilir bir şekilde götüren bir şey arıyoruz.
   * Hedefi bulmak için izleyici mermileri kullanın.
   * Avantajları
     + Kullanıcılar bir şeyin erken çalıştığını görebilir.
     + Geliştiriciler çalışmak için bir yapı oluşturur.
     + Bir entegrasyon platformunuz vardır.
     + Gösterecek bir şeyiniz vardır.
     + İlerlemeyi daha iyi gözlemlersiniz.
   * İzli mermiler her zaman hedeflerini vurmaz.
   * Bir prototip ile nihai sistemin belirli yönlerini keşfetmeyi hedefliyorsunuz.
   * İzleme kodu, uygulamanın bir bütün olarak nasıl bir arada kaldığını bilmek için kullanılır.
   * Prototip oluşturma, tek kullanımlık kod üretir.
   * İzleme kodu yalındır ancak eksiksizdir ve nihai sistemin iskeletinin bir parçasını oluşturur.
6. Prototipler ve Post-it Notlar
   * Riski değerlendirmek, ortaya çıkarmak ve önemli ölçüde daha düşük maliyetle düzeltme fırsatları sağlamak için yazılım prototipleri oluşturuyoruz.
   * Detaylardan vazgeçemediğiniz bir çevrede kendinizi bulmuşsanız, kendinize gerçekten prototip oluşturmaya ihtiyacınız olup olmadığını sormalısınız.
   * Prototiplenecek Şeyler
     + Mimari.
     + Var olan sistemdeki yeni özellikler.
     + Dış verinin yapısı veya içeriği.
     + Üçüncü parti araçlar veya bileşenler.
     + Performans konuları.
     + Kullanıcı arayüz tasarımı.
   * Öğrenmek için prototipleyin
   * Ayrıntılardan kaçının
     + Doğruluk.
     + Bütünlük.
     + Sağlamlık.
     + Stil.
   * Prototipleme Mimarisi
     + Ana bileşenlerin sorumlulukları iyi tanımlanmış ve uygun mu?
     + Ana bileşenler arasındaki iş birlikleri iyi tanımlanmış mı?
     + Bağlantı en aza indirildi mi?
     + Potansiyel çoğaltma kaynaklarını belirleyebilir misiniz?
     + Arayüz tanımları ve kısıtlamaları kabul edilebilir mi?
     + Her modülün yürütme sırasında ihtiyaç duyduğu verilere erişim yolu var mı?
   * Uygun şekilde kullanılırsa, prototip, çok fazla zaman, para, acı ve emek gibi etkenlerden sizi kurtarabilir.
7. Alan(Domain) Dilleri
   * Dilin sınırları, kişinin dünyasının sınırları kadardır. ( Ludwig Wittgenstein)
   * Problem alanına yakın programlama yapın.
8. Tahmin
   * Sürprizlerden kaçınmak için tahmin edin.
   * Ne kadar doğru, yeterince doğru mu?
     + Yüksek doğruluğa mı ihtiyaçları var yoksa kaba hesap mi arıyorlar?
     + Zaman tahminlerini doğru şekilde ölçeklendirin.

Application, table

Description automatically generated

* + - Tahminler nereden geliyor?
    - Geçmişte benzer bir durumda olan birine sorun.
    - Ne sorulduğunu anlayın.
    - Sistemin bir modelini oluşturun.
    - Modeli bileşenlere ayırın.
    - Her parametreye bir değer verin.
    - Cevapları hesaplayın.
    - Tahmin becerilerinizi takip edin.
  + Proje Takvimlerini Tahmin Etme
    - Bir projenin zaman çizelgesini belirlemenin tek yolu, aynı projede deneyim kazanmaktır. Aşağıdaki adımları tekrarlayarak artımlı geliştirme uygulayın:
      * Gereksinimleri kontrol edin.
      * Riski hesaplayın.
      * Tasarlayın, uygulayın ve entegre edin.
      * Kullanıcılarla beraber doğrulayın.
  + Programınızı kodla yineleyin.
  + Birisi sana bir tahminde bulunmanı istediğinde ne diyeceksin?
    - Size geri döneceğim.
  + Tahminlerinizin kaydını tutmaya başlayın. Her biri için ne kadar isabetli olduğunuzu takip edin. Hatanız %50'den büyükse, tahmininizin nerede yanlış gittiğini bulmaya çalışın.

Bölüm 3-Temel Araçlar

* Araçlar yeteneklerinizi artırır.
* Daha iyi araçlara sahip olursanız, onları daha iyi kullanmayı bilirsiniz ve daha üretken olursunuz.

1. Düz Metnin Gücü
   * Bilgiyi düz metinde tutun.
   * Metnin Gücü
     + Eskimeye karşı sigortalıdır.
     + Mevcut araçlardan yararlanılabilir.
     + Daha kolay test edilebilir.
2. Shell Games (Kabuk Oyunları - Terminal)
   * Terminalin gücünü kullanın
   * Terminale aşinalık kazanın ve eninde sonunda üretkenliğinizin arttığını göreceksiniz.
3. Güçlü Düzenleme
   * Editörde akıcı olun.
   * “AKICI” ne demek?
     + Metin düzenlerken karakter, kelime, satır ve paragraf bazında seçim yapabilmek.
     + Değişikliklerin ardından kodun girintisini ayarlayın.
     + Bir kod parçasını yoruma alın ya da çıkarın.
     + Değişikliği geri alın ya da ileri götürün.
     + Belirli bir satır numarasına gidin.
     + Editörünüzü belli pencerelere bölün.
     + Proje içerisinde arama yapın.
     + Derleme hatalarını gösterin.
     + Projenin testlerini çalıştırın.
     + Seçili satırları sıralayın.
4. Versiyon Kontrol
   * Her zaman versiyon kontrol sistemi kullanın.
5. Hata Ayıklama
   * Hata ayıklamanın sadece problem çözmek olduğu gerçeğini benimseyin ve ona bu şekilde saldırın.
   * Hata ayıklamanın psikolojisi
     + Suçu değil sorunu çözün.
   * Hata ayıklama zihniyeti
     + Panik yapmayın.
   * Nereden başlamalısınız?
     + Başlamadan önce kodun hatasız ve uyarısız bir şekilde derlenebildiğinden emin olun.
     + Herhangi bir sorunu çözmeye çalışırken, ilgili tüm verileri toplamanız gerekir.
   * Hata ayıklama stratejileri
     + Bir hatayı düzeltmeye başlamanın en iyi yolu onu tekrarlanabilir kılmaktır.
     + Kodu düzeltmeden önce başarısız test yap.
     + Hata mesajını okuyun.
     + Varsaymayın, kanıtlayın.

1. Metin Manipülasyonu
   * Bir metin manipülasyon dili öğrenin.
2. Mühendislik Günlükleri
   * Günlük tutmanın 3 temel yararı vardır
     + Hafızanızdan daha güvenilirdir.
     + Eldeki görevle alakalı olmayan fikirleri depolamak için bir yer sağlar.
     + Bir tür lastik ördek görevi görür.( Bir not almaya başlayabilir ve sonra aniden yaptığınız şeyin, notun konusunun tamamen yanlış olduğunu fark edebilirsiniz)

Bölüm 4- Pragmatik Paranoya

* + Mükemmel Yazılım Yazamazsınız

1. Sözleşme ile tasarım
   * Meyer ‘in fonksiyonlardan beklenenleri anlamak için göz önünde bulundurmamız gereken maddeleri:
     + Ön koşullar
     + Son koşullar
     + Sınıf değişkenleri
   * Ne olursa olsun, sözleşmeye uymamanın bir hata olduğu konusunda hemfikir olun.
   * Sözleşme ile tasarlayın.
   * Başlamadan önce neleri kabul edeceğiniz konusunda katı olun ve karşılığında mümkün olduğunca az söz verin.
2. Ölü programlar yalan söylemez.
   * Tüm hatalar size bilgi verir.
   * Kendinizi hatanın olamayacağına ikna edebilir ve görmezden gelmeyi seçebilirsiniz.
   * Bunun yerine pragmatik programcılar kendilerine bir hata varsa, çok, çok kötü bir şey olduğunu söylerler.
   * Hata mesajını okumayı unutmayın.
   * Ölü bir program normalde sakat bir programdan çok daha az hasar verir.
3. İddialı Programlama
   * İmkansızı Önlemek İçin İddiaları Kullanın
   * Performans sorunlarınız olsa bile, yalnızca sizi gerçekten etkileyen kontrolleri kapatın.
4. Kaynaklar nasıl dengelenir?
   * Başladığınız işi bitirin.
   * Kaynakları, tahsis ettiğiniz sıranın tersi sırayla serbest bırakın.
   * Bu şekilde, bir kaynak diğerine referanslar içeriyorsa kaynakları tek bırakmazsınız.
   * Aynı kaynak grubunu kodunuzda farklı yerlere tahsis ederken, onları her zaman aynı sırayla tahsis edin.
   * Bu, kilitlenme olasılığını azaltacaktır.
   * Bir kaynağı kim tahsis ederse, onu serbest bırakmaktan sorumlu olmalıdır.
5. Farlarınızı Geçmeyin
   * Daima küçük adımlar atın.
   * Geri bildirimleri kontrol edin ve ayarlamalar yapın.
   * Asla "çok büyük" bir adım veya görev almayın.
   * Geri bildirim ile tam olarak ne demek istiyoruz?
     + REPL’deki sonuçlar, API’leri ve algoritmaları anlamanız hakkında geri bildirim sağlar.
     + Birim testleri, son kod değişikliğiniz hakkında geri bildirim sağlar.
     + Kullanıcı demosu ve konuşma, özellikler ve kullanılabilirlik hakkında geri bildirim sağlar.
   * Çok büyük bir görev nedir?
     + Aylar sonraki tamamlanma tarihlerini tahmin etme.
     + Gelecekteki bakım veya genişletilebilirlik için bir tasarım planlayın.
     + Kullanıcının gelecekteki ihtiyaçlarını tahmin edin.
     + Gelecekteki teknoloji kullanılabilirliğini tahmin edin.