PRAGMATİK PROGRAMCI

Pragmatik programcı ile programcının farkı nedir? Eğer bir pragmatik programcıysanız büyük pencereden bakmayı bilirsiniz. Pragmatik programcı, pragmatik felsefenin temel taşlarından biri olan sorumluluk almayı kendine görev olarak edinir. Çünkü sorumluluk aldığında bu işin doğru olduğuna dair taahhütte bulunmuş olur. Herhangi bir hata sonucunda da dürüstçe hatasını kabul edip alternatif çözümler için fikir belirtir. Bu felsefeye göre müthiş bir planla hazırlanan projeler için bile her an her şey olabilir anlayışı geçerlidir. Fakat hata gerçekleştiğinde de başkalarına suç atarak bu işin içinden sıyrılmaya çalışmamalısınız. Bahaneler sizi pragmatik kılmaz. Mazeretleri sunmadan önce kendi kendinize bunların saçma olup olmadığı hakkında düşünmeniz ve eğer saçma değilse durumu düzeltmek için seçenekleri üstlerinize sunmanız gerekir. Yazılım entropisine gelecek olursak bu konu aslında çok basit bir örnek ile anlatılabilir. Kırık bir pencere. Kırık bir pencereyi üzerinde çalıştığınız veya sonradan dahil olduğunuz bir projedeki basit bir hata olarak görebilirsiniz. Bu basit hatayı çözmediğiniz sürece bu durum gitgide büyüyerek tıpkı kırık bir penceresi olan bir binanın çürümesi gibi sizin projenizin çürümesine sebep olacaktır. Bunun için yaptığınız işi mutlaka eksiksiz yapın.

- Taş Çorbası

Savaştan dönen ve aç olan 3 asker bir köye varır ve bir şeyler yemek için köylülerden yardım isterler. Köylüler de sıkıntıda oldukları için askerlere yemek vermeyi reddederler. İçlerinden bir asker bir taş alır ve su ile kaynatmaya başlar. Merak eden köylüler ne yapıyorsun diye sorar. Asker taş çorbası yaptığını söyler. Köylü sadece bu malzemelerle mi? Diye sorar. Asker, evet ama biraz havuç olsa daha iyi olurdu diye ekler. Köylüler hemen havuç getirir. Her bir malzeme getirdiklerinde asker şu olsa bu da olsa daha iyi olur diye söyler ve köylüler de hemen getirir. Sonunda birçok malzemeleri olur ve ziyafet çekerler. Hikâyenin burada anlatmak istediği askerin katalizör görevi gördüğü ve sonuç olarak herkesin kazanmış olmasıdır. Başarmak için önce insanlara bir şeyler gösterip onları etkilemelisin. İnsanlar devam eden bir başarıya katılmayı daha kolay buluyor.

Kullanıcılarınız ve gelecekteki geliştiriciler için, yeterince iyi bir yazılım sağlamak amacıyla kendimizi disipline etmeliyiz. Daha kısa inkübasyon süresi nedeniyle programlarımızın daha iyi olduğunu farkedebiliriz. Kullanıcılarımızı takasa dahil etmeliyiz. Çünkü bugünün mükemmel yazılımı genellikle yarının mükemmel yazılımına tercih edilir. Aşırı ayrıntılandırma kodu mahvedeceğinden dolayı kodlamayı ne zaman durdurmamız gerektiğini bilmeliyiz.

Bilgi ve deneyimimiz en değerli mesleki varlığımızdır. Ancak varlıklarının süresi doluyor. Bilgi portföyünüzü çeşitlendirilmiş ve güncel tutmalıyız. Düzenli olarak yatırım yapmalı çeşitlendirmeli ve riski yönetmeye çalışmalıyız. Gözden geçirmek ve yeniden dengelemek için zaman ayırmalıyız.

Sorun, tüm kod tabanında bulunan şeylerin bir temsilini değiştirmeniz gerektiğinde ortaya çıkar. Her bilgi parçası, bir sistem içinde tek, açık, yetkili bir temsile sahip olmalıdır.

Birindeki değişiklikler diğerlerini etkilemiyorsa iki veya daha fazla şey ortogonaldir. Kohezyon da denir. "Utangaç" kodu yazın.

Faydaları:

- Verimlilik Kazanın
- Riski azaltmak
- Proje Ekipleri: İşlevsellik bölünmüştür
- Tasarım: Komple bir projeyi bileşenleri aracılığıyla tasarlamak daha kolay
- Araç Setleri ve Kitaplıklar: Ortogonalimi korumak için akıllıca seçim yapın
- Kodlama: Kod eklerken dikeyliği korumak için şunları yapın:
- Test etme: Ortogonal sistemlerin test edilmesi daha kolaydır.
- Dokümantasyon: Ayrıca kalite kazanın

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.

Riski analiz etmek ve ortaya çıkarmak ve büyük ölçüde azaltılmış bir maliyetle düzeltme şansı sunmak için yazılım prototipleri oluşturuyoruz.

Bilgiyi düz metin olarak tutun. Komut kabuklarının gücünü kullanın. Her Zaman kaynak kodu denetimini Kullan. Sorunu düzeltin suçluyu aramayın ve panik yapmayın. "Ama bu olamaz" diye başlayan düşünce zincirinde tek bir nöronu boşa harcamayın çünkü oldukça açık bir şekilde olabilir ve olmuştur. Bir sorunun sadece bu özel görünümünü değil, kök nedenini keşfetmeye çalışın.

Bir Metin işleme dili öğrenin.

Mükemmel yazılım yazamazsınız. Kısa bilgi işlem tarihinde hiç kimse mükemmel bir yazılım parçası yazmadı. Pragmatik Programcılar da kendilerine güvenmezler.

Doğru bir program, iddia ettiğinden daha fazlasını veya daha azını yapmayan programdır.

"Tembel" kodu yazın: Başlamadan önce neyi kabul edeceğiniz konusunda katı olun ve karşılığında mümkün olduğunca az söz verin.

Tüm hatalar size bilgi verir. Pragmatik Programcılar kendilerinde bir hata varsa bunun kötü bir şey olduğunu söylerler. Kodunuz, imkansız olması gereken bir şeyin gerçekleştiğini keşfettiğinde, programınız artık uygulanabilir değildir.

Kaynakları yönetirken: bellek, işlemler, iş parçacıkları, uçar, zamanlayıcılar sınırlı kullanılabilirliğe sahip her türlü şey, işimiz bittiğinde bunları kapatmamız, bitirmemiz, silmemiz, yeniden yerleştirmemiz gerekir.

Diğer kaç modülle etkileşime girdiğinize ve onlarla nasıl etkileşime girdiğinize dikkat edin. Nesneler arasındaki ilişkileri doğrudan çaprazlamak, hızlı bir şekilde kombinatoryal patlamaya yol açabilir.

The Law of Demeter kullanmak, kodunuzu daha uyarlanabilir ve sağlam hale getirecektir, ancak bunun bir bedeli vardır: talebi basitçe bir temsilciye ileten çok sayıda sarmalayıcı yöntem yazacaksınız. Hem çalışma zamanı maliyeti hem de alan yükü getirir. Özel uygulamanız için artıları ve eksileri dengeleyin.

"Ayrıntıları açıkla!" Onları koddan çıkarın. Hazır gelmişken, kodumuzu son derece yapılandırılabilir ve "yumuşak", yani değişikliklere kolayca uyarlanabilir hale getirebiliriz.

Uygulamayı mümkün olduğunca meta veriler aracılığıyla yapılandırmak ve sürmek istiyoruz. Genel durum için programlayın ve ayrıntıları derlenmiş kod tabanının dışında başka bir yere koyun.

Çalışırken yapılandırmalarını yeniden yükleyebilen programlar yazmak esnek bir yaklaşımdır.

Nesneler, yalnızca ihtiyaç duydukları olayları almak için kaydolabilmeli ve ihtiyaç duymadıkları olaylar asla gönderilmemelidir. Çok önemli bir tasarım konseptini uygulamak için bu yayınlama/abone olma mekanizmasını kullanın: bir modelin model görünümlerinden ayrılması.

Model-Görünüm-Denetleyici

Modeli hem onu temsil eden GUI'den hem de görünümü yöneten kontrollerden ayırır.

Avantajları:

- Aynı veri modelinin birden çok görünümünü destekleyin.
- Birçok farklı veri modelinde ortak görüntüleyiciler kullanın.
- Geleneksel olmayan giriş mekanizmaları sağlamak için birden çok denetleyiciyi destekleyin.

Denetleyici daha çok bir koordinasyon mekanizmasıdır ve herhangi bir girdi aygıtıyla ilişkili olması gerekmez.

- Model hedef nesneyi temsil eden soyut veri modeli. Modelin herhangi bir görünüm veya denetleyici hakkında doğrudan bilgisi yoktur.
- Görünüm Modeli yorumlamanın bir yolu. Modeldeki değişikliklere ve denetleyiciden gelen mantıksal olaylara abone olur.
- Denetleyici Görünümü denetlemenin ve modele yeni veriler sağlamanın bir yolu. Olayları hem modele hem de görünüme yayınlar.

Bir karatahta sistemi, bilgi tüketicilerinin ve üreticilerin anonim ve eş zamansız olarak veri alışverişinde bulunabileceği bir forum sağlayarak, nesnelerimizi birbirinden tamamen ayırmamıza olanak tanır.

Şansa güvenerek tesadüfen programlama yapmaktan ve kasıtlı olarak programlama lehine tesadüfi başarılardan kaçınmalıyız.

Kasıtlı Olarak Nasıl Programlanır:

- Her zaman ne yaptığınızın farkında olun.
- Gözünüz kapalı kod yazmayın.
- Bir plandan ilerleyin.
- Yalnızca güvenilir şeylere güvenin.
- Varsayımlarınızı belgeleyin.
- Sadece kodunuzu test etmeyin, aynı zamanda varsayımlarınızı da test edin.

Pragmatik Programcılar, algoritmaların kullandığı kaynakları, zamanı, işlemciyi, belleği vb. tahmin eder. Uygun algoritmaları seçme konusunda pragmatik olun; en hızlısı her zaman iş için en iyisi değildir. Erken optimizasyon konusunda dikkatli olun. İyileştirmeye zaman ayırmadan önce bir algoritmanın gerçekten bir darboğaz olduğundan emin olun.

Kodun gelişmesi gerekiyor; statik bir şey değil.

Ne Zaman Yeniden Düzenleme Yapmalısınız:

- Çoğaltma DRY ilkesinin ihlal edildiğini keşfettiniz
- Ortogonal olmayan tasarım. Daha ortogonal hale getirilebilecek bazı kodlar veya tasarımlar keşfettiniz.
- Eski bilgi. İşler değişir, gereksinimler değişir ve sorunla ilgili bilginiz artar. Kodun yetişmesi gerekiyor.
- Verim. Performansı artırmak için işlevselliği sistemin bir alanından diğerine taşımanız gerekir.

Test edilebilirliği en baştan yazılıma dahil edin ve her bir parçayı birbirine bağlamaya çalışmadan önce baştan sona test edin. Davranışını doğrulamak için her modül üzerinde ayrı ayrı testler yapılır. Bir yazılım birimi testi, bir modülü çalıştıran koddur.

Test kodunu kolayca erişilebilir hale getirerek, kodunuzu kullanabilecek geliştiricilere iki paha biçilmez kaynak sağlamış olursunuz:

- Modülünüzün tüm işlevlerinin nasıl kullanılacağına ilişkin örnekler
- Kodda gelecekte yapılacak değişiklikleri doğrulamak için regresyon testleri oluşturmanın bir yolu

Bir sihirbaz kullanırsanız ve ürettiği kodun tamamını anlamazsanız, kendi uygulamanızın kontrolü sizde olmayacaktır. Anlamadığınız sihirbaz kodunu Kullanmayın.

Gereksinimler mimari değildir. Gereksinimler tasarım veya kullanıcı arayüzü değildir. Gereksinimler ihtiyaçtır.

Kullanıcıların ve geliştiricilerin aynı şeyi farklı isimlerle andığı veya daha da kötüsü farklı şeylere aynı isimle atıfta bulunduğu bir projede başarılı olmak çok zordur.

Bulmacaları çözmenin anahtarı, hem üzerinize konan kısıtlamaları tanımak hem de sahip olduğunuz özgürlük derecelerini tanımaktır, çünkü bunlarda çözümünüzü bulacaksınız.

Çözümü bulamıyorsanız, geri çekilin ve kendinize şu soruları sorun:

- Daha kolay bir yolu var mı?
- Doğru sorunu çözmeye mi çalışıyorsunuz, yoksa çevresel bir teknik ayrıntıyla dikkatiniz mi dağıldı?
- Bu şey neden bir sorun?
- Çözmeyi bu kadar zorlaştıran şey nedir?
- Bu şekilde mi yapılması gerekiyor?
- Hiç yapılması gerekiyor mu?

Yazmaya başlamak için otururursanız ve aklınızda rahatsız edici bir şüphe varsa, kulak verin. Rahatsız edici şüpheleri dinleyin hazır olduğunuz zaman başlayın.

Resmi yöntemlerin kölesi olmayın.

Biçimsel yöntemlerin bazı ciddi eksiklikleri vardır:

- Diyagramlar son kullanıcılar için anlamsızdır, kullanıcıya bir prototip gösterin ve onunla oynamasına izin verin.
- Resmi yöntemler uzmanlaşmayı teşvik ediyor gibi görünüyor. Bir sistemin her yönünü derinlemesine kavramak mümkün olmayabilir.
- Çalışma zamanında uygulamaların karakterini değiştirmemize izin vermek için meta verileri kullanarak uyarlanabilir, dinamik sistemler yazmayı seviyoruz, ancak mevcut resmi yöntemlerin çoğu buna izin vermiyor.

Bir bireyin takımlar için çalışmasına yardımcı olan pragmatik teknikler. Kalite bir ekip meselesidir. Bir bütün olarak ekipler, kimsenin düzeltmediği küçük kusurlara, kırık camlara müsamaha göstermemelidir. Kalite ancak tüm ekip üyelerinin bireysel katkılarından gelebilir.

İnsanlar, bir başkasının bir sorunla ilgilendiğini veya kullanıcınızın talep ettiği bir değişikliği ekip liderinin onaylamış olması gerektiğini varsayar. Bununla savaş.

Bir varlık olarak ekibin dünyanın geri kalanıyla açık bir şekilde iletişim kurması gerekir. İnsanlar, herkesi iyi hissettirecek, iyi hazırlanmış bir performans izleyeceklerini bildikleri için onlarla buluşmayı sabırsızlıkla bekliyorlar. Ekiplerin tek olarak iletişim kurmasına yardımcı olan basit bir pazarlama hilesi vardır: Bir marka oluşturmak.

Otomasyon, her proje ekibinin temel bir bileşenidir.

Manuel Prosedürleri Kullanmayın.

Cron'u kullanarak, yedeklemeleri, gecelik derlemeleri, Web sitelerini planlayabiliriz... katılımsız, otomatik olarak. Tek bir komutla kontrol etmek, inşa etmek, test etmek ve göndermek istiyoruz.

Derleme, boş bir dizini alan ve projeyi sıfırdan oluşturan, nihai çıktı olarak üretmeyi umduğunuz her şeyi üreten bir prosedürdür.

- Depodaki kaynak kodunu kontrol edin.
- Projeyi sıfırdan oluşturun.
- Dağıtılabilir bir görüntü oluşturun.

- Belirtilen testleri çalıştır.

Bilgisayarın tekrarlayan, sıradan olanı yapmasına izin verin, bizden daha iyi bir iş çıkaracaktır. Yapacak daha önemli ve daha zor işlerimiz var.

Pragmatik Programcılar hatalarımızı şimdi bulmaya yönelirler, bu nedenle başkalarının hatalarımızı daha sonra bulmasının utancına katlanmak zorunda kalmayız.

Erken Test Edin. Sık Test Edin. Otomatik Olarak Test Edin.

- Her derlemede çalışan testler en etkili olanlardır.
- Bir hata ne kadar erken bulunursa, düzeltilmesi o kadar ucuz olur. "Biraz kodla, biraz test et".

Ne Test Edilmeli:

- Birim testi: bir modülü çalıştıran kod.
- Entegrasyon testi: projeyi oluşturan ana alt sistemler birbirleriyle iyi çalışır ve çalışır.
- Doğrulama ve doğrulama: Kullanıcıların ihtiyaçlarını karşılayıp karşılamadığınızı test edin.
- Kaynak tükenmesi, hatalar ve kurtarma: gerçek dünya koşullarında nasıl davranacağını keşfedin. (Bellek, Disk, CPU, Ekran…)
- Performans testi: gerçek dünya koşullarında performans gereksinimlerini karşılar.
- Kullanılabilirlik testi: gerçek kullanıcılarla, gerçek çevre koşullarında gerçekleştirilir.

Nasıl Test Edilir:

- Regresyon testi: mevcut testin çıktısını önceki (veya bilinen) değerlerle karşılaştırır. Testlerin çoğu regresyon testleridir.
- Test verileri: Yalnızca iki tür veri vardır: gerçek dünya verileri ve sentetik veriler.
- Çalıştırma GUI sistemleri: basit bir olay yakalama/oynatma modeline dayalı özel test araçları gerektirir.
- Testleri test etme: Belirli bir hatayı tespit etmek için bir test yazdıktan sonra, hataya kasıtlı olarak neden olun ve testin şikâyet ettiğinden emin olun.

Bir tutarsızlık varsa, daha iyi veya daha kötü olması için önemli olan koddur.

Genel olarak, yorumlar bir şeyin neden yapıldığını, amacını ve hedefini tartışmalıdır.

Siz (ve sizden sonrakiler) kodu yüzlerce kez okuyacağınızı, ancak yalnızca birkaç kez yazacağınızı unutmayın.

Anlamsız isimlerden daha da kötüsü yanıltıcı isimlerdir.

Kaynak dosyada görünmesi gereken en önemli bilgi parçalarından biri yazarın adıdır; dosyayı en son kimin düzenlediği değil, sahibinin adıdır.

Web'de çevrimiçi olarak yayınlayın.

Her Web sayfasına bir tarih damgası veya sürüm numarası koymayı unutmayın.

Bir işaretleme sistemi kullanarak, ihtiyaç duyduğunuz kadar farklı çıktı formatı uygulama esnekliğine sahip olursunuz.

Bir projenin başarısı, kullanıcılarının beklentilerini ne kadar iyi karşıladığı ile ölçülür. Kullanıcılarınızın beklentilerini yavaşça aşın.

Pragmatik Programcılar sorumluluktan kaçmazlar. Bunun yerine, zorlukları kabul etmekten ve uzmanlığımızı iyi bilinir kılmaktan mutluluk duyuyoruz.

Sahip olmanın gururunu görmek istiyoruz. "Bunu ben yazdım ve işimin arkasındayım."