* Debug yapiyorsan, kodu anlamadin demektir.
* Debug yaparken kodu anlamiyorsan, nasil debug yapilir, onu anlamadin demektir.
* Debug yaparken kodu anladigini dusunuyorsan, aslinda debuga ihtiyacin olmadigini anladin demektir.
* Debug etmek kodun mental bir modelinin eksikligine isarettir..

İki durumda debugger gerekli olabilir:

* Mevcut kod devralındığında.
* Yeni kod yazıldığında.

Ben her iki durumda da debugger çalıştırmamaya dikkat edilmeli. Amaç sıfır debug aktivitesi ile uygulamayı istenilen seviyeye getirebilmek. Peki neden?

Öncelikle mental modelin ne olduğu tanımı ile başlayalım. Mental model kısaca kodun programcının kafasındaki karşılığıdır. Bu model ne kadar net ise, debug işlemi o oranda önemini ve gerekliliğini yitirecektir. Özellikle programcı yeni bir kod birimi geliştirirken kesinlikle ne yapmak istediğinin bir modelini kafasında oluşturmuş olmalıdır. Mental modelin karşılığı çoğu zaman uml diyagramlarında yer alan nesne ve sınıfsal ilişkilerdir. Bunun yanı sıra mental model kullanılan veri yapılarını, algoritmaları, uygulamanın davranış biçimini ve gerekli çözümleri ihtiva eder. Oturmuş olan bir mental modeli çalışan koda dönüştürmek sadece kodlama aktivitesidir. Mental model kodlama işleminin ya da implementasyonun hangi yöne doğru gitmesi gerektiğini tayin ederken, kodun doğruluğu ise oluşturulan testler tarafından kontrol edilir. Bu ikisinin kombinasyonu debug işlemini tamamen gereksiz kılacaktır.

Debugger kullanımı mental modelin yokluğuna, tam oturmamış olmasına ya da doğru olmadığına bir işarettir. Kod yazarken debug işlemine başladığım anda normal olmayan bir durum oluştu demektir. Kod birimleri bir uygulamanın davranış biçimi tayin ederler. Yeni oluşturulan kod birimlerinde uygulamanın nasıl bir davranış gösterdildiği ilk önce debug oturumunda keşfediliyorsa, bu zaten programcının tam anlamıyla ne yapmak istediğinden ve nerede olduğundan bihaber olduğu anlamına gelmektedir, çünkü debug ederek ilerleyen programcı bir şekilde kodu adapte ederek, istediği yere varmaya çalışacaktır. Programcı implementasyona başlamdan önce problemin çözümü hakkında yeterince düşünce sarf etmemiş ve bu işlemi debug oturumuna bırakmıtır. Yani aslında bir davranış biçimini (behaviour) oluşturmaktan sorumluyken, programcı debug ederek bir davbranış biçimi keşfetmiştir ve çözümü buradan ilerleyerek bulmuştur. Aslında tam tersi olmalıdır. Programcı mental modelin öngördüğü çözümü implemente etmelidir. Bu şekilde elinde bir çözüm olmadan yola çıkmış ve anlamadığı yarım yamalak bir implementasyonu debug ederek, çözüme ulaşmaya çalışmıtır. Çözüme ulaşmış olması onun başarlı olduğu anlamına gelmez. Bu daha ziyade programcının problemler hakkında daha yüksek bir seviyede düşünce sarf etmediği, sistemden nasıl bir davranış beklediğini bilmediği, aklına gelen ilk fikir ile implementasyonu oluşturduğu ve çıkmaza girdiğince hemen debuggera sarıldığı anlamına gelmektedir.

Debug ederken kodun belli bir satırının ne yaptığı hiç bir anlam taşımamaktadır. Programcının bunu bilmesi bir şeyi değiştirmez. Asıl önemli olan nokta, programcının bilinçli bir şekilde mental modelin öngördüğü çözümü implemente etmesidir. Bunun için kesinlikle debug oturumuna ihtiyacı yoktur. Oluşturduğu testler aracılığı ile [kodu sürekli yoğurarak](http://www.kurumsaljava.com/2012/08/09/yazilimda-ceviklik-iflas-mi-etti/), istediği çözümü ve gerekli olan davranışı implemente edebilir. Burada önemli olan soyut çözümü somutlaştırıp, koda dökmektir.

Debugger seviyesine inmiş bir programcının kafası detaylarla meşgul olacak ve mevcut problemi bir şekilde çözmeye çalışacaktır. Aslında koda mental model aracılığı ile daha yüksek bir seviyeden bakması, onun daha soyutlayıcı çalışmasını ve problemi daha yüksek bir seviyede genelleyerek, çözmesini kolaylaştıracaktır. Detaylarda kaybolmuş bir beyin soyutlamaktan acizdir ve genel geçerli çözümler üretemez. Bu sadece daha yüksek bir seviyede yapılabilecek bir aktivitedir. Beynin genel resmi görebilmesi için detaylardan uzak durması gerekmektedir. Ne yazık ki debugging işlemi bunu sağlayamamakla birlikte, daha da kötüleştirmektedir.

Ben çözümleri dağ yamaçlarındaki patikalara benzetirim. Bu patikalar zaman içinde kullanıcılarının ayak izleriyle oluşmuş yapılardır. Her yeni bir kullanıcısı ile patikanın yapısı daha belirgin hale gelir. Her yeni ayak izi onun patika olmasını ve patika olarak kalmasını kolaylaştırır. Debug süreci de beyinde patikalar oluştur. Hatayı bulmak ya da kodu anlamak için aynı satırların tekrar, tekrar debug edilmesi kodun öngördüğü çözümü programcının beyninde kemikleştirir. Beyin bunun haricinde bir çözümün varlığını kabul edemez hale gelir. Bu sebepten dolayı programcının çözüme ulaştırmayan bir kod parçasını kaldırıp, atması ve yeniden implemente etmesi çok zor ya da imkansızdır.

Buraya kadar olan bölümde yazılan yeni kod birimleri için neden debugger kullanılmaması gerektiği konusuna değinmeye çalıştım. Bazı okurlarım haklı olarak debug etme aktivitesinin devralınan kodlarda gerekli olduğu fikrine sahip olabilirler. Ben burada da aynı fikirde değilim. Nedenini açıklamaya çalışayım.

Debug işlemi zaman kaybından başka bir şey değildir. Hata bulmak ya da kodu anlamak için kullanılması bu sebepten dolayı maliyeti artırır. Sonuç itibarı ile programcının belli bir zamanı vardır ve bu zaman diliminde neler yapabileceği bellidir. Programcı saatler ya da günler harcayarak, kodu debug edebilir ve anlamaya çalışabilir. Bu geride kodu anladığını zannettiği sübjektif bir histen başka bir şey bırakmayacaktır. Doğal olarak burada sormamız gereken soru şu olmalıdır? Debug ederek programcının eline ne geçmiştir? Bunun yerine programcı kodu okuyarak ve uzun metotları parçalarına bölerek, kodu daha okunur bir hale getirebilir. Bu yeniden yapılandırma işlemini mevcut testler destekleyebilir ya da programcı yeni testler oluşturabilir. Programcı kodu anlama sürecini yeni log satırmaları oluşturarak, kolaylaştırabilir. Bu sayede programcı daha verimli bir çalışma sergilemiş olur.

Debugging işlemi teknik açıdan bir ilüzyondur. Modern mikro işlemciler “out of order execution” olarak bilinen mekanizma ile çalışırlar. Bu işlemcinin kod birimlerini programcının bilmediği bir sırada koşturabileceği anlamına gelmektedir. Bunun yanına bir de derleyicilerin yaptıkları optimizasyonlar ve Java dilinde olduğu gibi “sequential consistency” özelllikleri geldiğinde, programcının debug ettiği kod ile işlemcinin gördüğü kod aynı olmayacaktır. Bu sebepten dolayı birçok buğ için “benim bilgisayarımda çalışıyor” ifadeleri kullanılmaktadır.

Debugger kullanımının kötü bir alışkanlık olduğunu düşünüyorum. Programcı sahip olduğu sorumlulukları (kodu yeniden yapılandırma, test yazma, çözüm oluşturma) göz ardı etmekte ve genel resmi görmeksizin, detaylarda kaybolmayı göze almakadir. Oysaki debug işlemi ile lokal bir problemi çözmek yerine, oluşan hatanın gerçek kaynağını anlamaya çalışmak daha genel bir çözüm oluşturmayı mümkün kılacaktır.

Debugger en son çare olarak kullanılabilecek bir araçtır. Ben örneğin yeni öğrendiğim dillerde sıkça debugger kullanıyorum. Bu şekilde dilin yapısını ve çalışma tarzını anlamam kolaylaşıyor. Ya da bir algorıtma implemente ederken debugger kullandığım olmuştur. Debugger gerektiği yerde kullanılmalıdır. Ama bundan her zaman daha iyisi beynin kullanılmasıdır.