Yazılım Mühendisliği Lisans Programının Temel İlkeleri

Zeynep ALTAN
Yazılım Mühendisliği Bölümü
Beykent Üniversitesi
zeynepaltan@beykent.edu.tr

Özet

Farklı disiplinlerden bir grup bilim insanı 1968 yılında Almanya'da düzenledikleri ünlü NATO konferansında "Yazılım Mühendisliği" terimini ortaya atmışlar ve bunun çözülmesi gereken bir problem olduğunu ileri sürmüşlerdir. Böylece bu yeni terimin bilgisayar bilimlerinin bir alt disiplini olup olmadığı tartışılmaya başlamıştır. O günlerden beri devam eden tartışmaların doğruluğu, teknolojinin gelişmesi ile birlikte yazılım sektöründe görülen ilerlemelerle de onaylanmaktadır. Buna rağmen, yazılım mühendisliği hala kendisini betimlemeye ve diğer mühendislik disiplinleri arasında bir yer bulmaya çalışmaktadır.

1. Giriş

Yazılımın uygulama alanları genişledikçe bu disipline olan talep de hızla artmaktadır. Bu artışda İnternet kullanımındaki göz ardı edilemeyecek artış ile birlikte, bilgisayar programlarında büyük miktarda verinin kullanılması ve bunların işlenmesine olan gereksinimin etkisi de rol oynamaktadır. Ayrıca bilişsel makineler, grid hesaplama, biyolojik sensörler gibi yeni uygulama geliştirme alanları da yazılım mühendislerine olan gereksinimi hızla arttırmaktadır. Disiplinli ve sistematik olarak çalışan yazılım mühendislerinin temel ilkesi işin "zamanında ve minimum maliyetle" tamamlanarak, geliştirilen ürünün çalışmaya başlamasıdır.

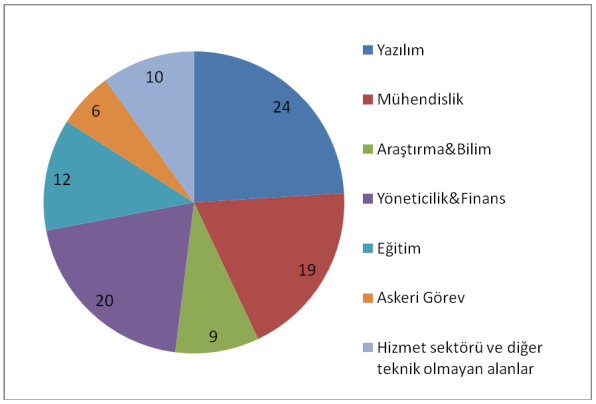
Yazılım mühendisliği eğitimi, müşterinin isterleri doğrultusunda analiz ve tasarım yapabilecek, bunları geliştirebilecek ve elde ettiği ürünü müşteriye teslim edebilir hale getirebilecek yeni bir mühendis tipinin eğitimi amacı ile başlatılmıştır. Bu alanda geliştirilen ürünler, diğer mühendislik disiplinlerinin ürünleri ile karşılaştırıldıklarında nitelik bakımından birbirlerinden çok farklı değildir. Fakat yazılım ürünleri diğer mühendislik ürünlerinde olduğu gibi imal edilmezler; sadece tasarlanabilirler. Her iki ürün türünün de yüksek kalite koşullarını kendi standartlarında mutlaka sağlamaları gerektiği doğaldır.

Bu açıklamalar doğrultusunda yazılım mühendisliği eğitiminin tasarım uzmanlığı ve profesyonel becerilerden, teknoloji ile zenginleştirilmiş sistem geliştirilmesine kadar oldukça geniş çerçevesinin olduğu söylenebilir. Yazılım sistemlerinin karmaşıklığı ve uygulandıkları ortamda çok hızlı yayılmaları, güvenli kod kullanımını ve yazılımın bazı temel ölçütleri sağlamasını gerektirir. Geleneksel yazılım mühendisliği programları yazılım sistemlerinin yaşam döngüsüne göre düzenlenmektedir. Bu da gereksinmeler analizi, programın çalışacağı ortamın ve sunacağı özelliklerin betimlenmesi; bir başka ifade ile belirtim, tasarım, yazılım sisteminin işlemeye başladığı geliştirme aşaması olarak özetlenebilir. Yazılımın geliştirme aşaması gerçekleştirim, test ve yazılım ürününün yüksek verimle çalışabilmesi için yapılan bakım çalışmalarını içerir.

2. Yazılım Mühendisliği Eğitimi Niçin Gereklidir?

Ülkemizde olduğu gibi dünyanın pek çok ülkesinde mühendislik ve fen programlarından mezun olan öğrenciler bir süre sonra yazılım geliştirmeye ilgi duymakta ve bu tür işlere yönelmektedirler. Maalesef bu kişilerin pek çoğu ya hiç bir yazılım eğitimi almamışlardır; ya da yazılım bilgileri minimum düzeydedir. American Institute of Physics Statistical Research Center [1] tarafından 2001 yılında yayınlanan Tablo 1'de verilen rapor bu alan değiştirme durumunu göstermektedir. Rapora göre fizik alanında lisans diplomasına sahip öğrencilerin %24'i mezuniyetlerinden 5 ile 7 yıl sonra yazılım sektöründe çalışmaya başlamışlardır. Bu kişilerin çoğunun eğitimleri süresince yazılım mühendisliği temel kavramları ile ilgili fazla bilgileri olmadığı bir gerçektir. Yazılım mühendisliği eğitiminin yaygınlaşması ile, diğer mühendislik ya da fen bilimleri alanlarından yazılım sektörünü seçenlerin daha ileri düzeyde bilgilenmeleri zorunluluk olmaktadır. Bunda hızla artan yazılım uygulama alanlarının, konusunda uzmanlaşmış kişilere olan gereksinimi arttırması da rol oynamaktadır.

Tablo 1: Fizik Mezunlarının Yöneldikleri İş Alanları



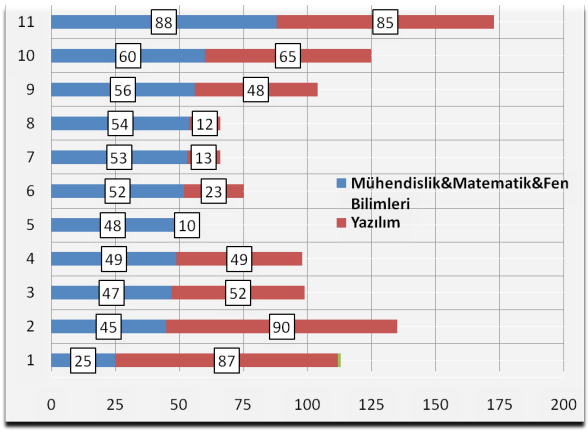
Diğer fen bilimleri dalları mezunları, hatta mühendislik dışı disiplinlerden kişiler de profesyonel yaşamlarında bir süre sonra yazılım geliştirmeye yönelebilmektedir. Bu kişilerin de lisans programlarını tamamlamadan önce, yazılım mühendisliği eğitiminin temel kavramları ile ilgili olarak bilgilenmiş olmaları zorunluluktur. Gerekli düzeyde benzeri eğitiminalınmış olması diğer tüm mühendislik ve fen bilimleri bölümleri için de geçerlidir.

Tablo 2'de ise yine 2001 yılı istatistiklerine göre fizik mezunlarının çalışma alanı olarak mühendislik, matematik ve fen bilimlerini ya da yazılım sektörünü seçmeleri durumunda, bilgi ve becerilerine ilişkin 11 farklı kritere göre değerlendirilmeleri sonuçları verilmektedir. Buna göre bilimsel olarak problemi çözebilme becerisi hem mühendislik ve fen bilimleri alanında çalışanlar, hem de yazılım sektöründe çalışanlar için önemlidir. Oysa mühendislik ve fen bilimleri alanında çalışanların yazılım geliştirme becerilerinin yazılım sektöründe çalışanlara göre üçte bir oranından daha az olduğu gözlenmektedir. Ayrıca yazılım sektörü çalışanlarının fizik mezunu olmalarına rağmen çok düşük düzeyde fizik bilgisine sahip olmaları ve fizik kurallarını uygulamaları gerekmektedir. Bilgiyi sentezleyebilme becerisinin her iki çalışma alanı için yaklaşık aynı oranda önemli olması da doğaldır.

Özet olarak; Tablo 1 ve Tablo 2 sadece bir meslek grubunun yazılım sektöründe uzmanlaşmayı tercih etmesi durumunu irdelemektedir. Bilgisayar ve yazılım mühendisliği dışındaki diğer meslek gruplarındaki kişiler de benzeri alan seçimi eğiliminde olduklarında minimum ölçekte bu sonuçları içereceklerdir.

Diğer bilim dallarından yazılım sektörünü seçenlerin incelenmesinden sonra, günümüzde çok miktarda verinin işlendiği cep telefonlarından büyük askeri sistemlere kadar yüksek teknoloji gerektiren problemlerin çözümü için, sonradan yazılım sektörüne geçenlerin bilgi ve becerilerinin yeterli olmayacağı gerçektir.

Tablo 2-a: İki Farklı İş Alanına göre Bilgi ve Beceriye İlişkin Parametrelerin Dağılımları



Tablo2-b Farklı İş Alanlarının Değerlendirildiği Parametreler

	Parametre Değerleri
1	Yazılım Geliştirme
2	Bilgisayar Programlaması
3	Ürün Tasarımı
4	Modelleme ve Benzetim
5	Fizik Bilgisi
6	Bilimsel Yazılım
7	Laboratuar ve Aygıt Kullanımı Becerileri
8	Fizik Kuralları
9	Matematiksel Beceriler
10	Bilginin Sentezlenmesi
11	Bilimsel Problem Çözme

Yazılım mühendisliği programının temelini bilgisayar bilimleri, matematik, fen ve doğa bilimleri oluşturur. Böyle bir yapı kurulduğunda, bölümün öğrencileri yaşam döngüsü süreçlerine uygun olarak ürün geliştirmesini öğrenecekler ve problemlerinin çözümünde nitelikli ürünler elde edebileceklerdir. Programlar problemlerin analizi, tasarımı ve modellenmesi ile birlikte, yazılım kalite özniteliklerini içerecek şekilde tasarlanmaktadır.

3. Yazılım Mühendisliği Programı Bilgisayar Bilimleri Programı mıdır?

Bu noktada "yazılım mühendisliği, bilgisayar bilimleri ya da Türkiye'de karşılığı olduğu şekilde bilgisayar mühendisliğinin bir alt disiplini midir?" sorusu sorulabilir. Soru kısaca: "elektrik mühendisliğinin bilimsel temeli nasıl fizikse, yazılım mühendisliğinin bilimsel temeli de bilgisayar bilimleri, ya da bilgisayar mühendisliğidir" şeklinde cevaplanabilir. Bir başka bakış açısına göre ise, yazılım mühendisliği inşaat mühendisliği, makine mühendisliği, kimya mühendisliği, elektrik-elektronik mühendisliği gibi farklı mühendislik disiplinlerini içeren kümenin bir elemanıdır [2].

Bilgisayar bilimleri ile yazılım mühendisliği arasındaki ilişki açıklanmadan önce "elektrik mühendisliği eğitimi öncelikli olarak fiziğe bağlı ise, niçin fizik bölümü ve elektrik mühendisliği olarak iki farklı program açılmaktadır? Elektrik mühendisleri niçin sadece fizik ile ilgilenmemektedir?" sorusunun cevaplanması uygun olacaktır.

Elektrik mühendisliği ile fizik programlarının uygulanma alanları birbirinden tamamen farklıdır. Çünkü her iki program farklı türde uzman yetiştirmektedir. Elektrik mühendisliği mezunlarının görevi başkalarının kullanımı için ürün tasarlanmaktır. Diğer uzmanlık alanı olan fizikte ise, araştırmalarla elde edilen bilgilerin bu alanda çalışan bilim insanları ile paylaşılması ve öğrenilenlerin sürekli olarak yenilenmesi temel hedeftir. Örneğin manyetik fizik ve fiziksel alan ya da kuantum fiziği ile ilgili yenilikler, mühendislik eğitim programlarına bunların çeşitli sektörlerdeki uygulamaları şeklinde yansır. Öğrenciler de ilgi alanlarına göre elektrik mühendisliğini seçtiklerinde problem çözümüne, yani yapılanmaya ilgi duyarlar; diğer gruptaki fizik öğrencileri ise problem çözmeden ziyade öğrenmeye ilgi duyarlar.

Benzeri açıklamalar yazılım mühendisliği ve bilgisayar bilimleri için de yapılabilir. Ayrıca bilgisayar bilimleri ve yazılım mühendisliğinin birbirini tamamladığı söylenebilir. Bu da yazılım mühendisliğinin hem hesaplama disiplini olarak bilgisayar bilimleri bakış açısından, hem de bir mühendislik disiplini bakış açısından tanımlanması demektir. Bilindiği gibi bilgisayar bilimlerinin temeli, hesaplama özelliklerinin incelenmesidir. Yazılım mühendisliğinin temeli ise, pratik olarak amaca ulaşmak için gerekli hesaplamaların tasarımıdır. Karmaşık yazılım sistemlerinin inşası için gerekli olan süreçler geliştirilen sistemin entegrasyonu, yazılım niteliğinin sağlanması ve yazılım projesinin yönetimi olarak özetlenebilir.

Kısaca, yazılım mühendisliği bir mühendislik dalı olarak bilgisayar bilimleri programlarından farklıdır. Derleyiciler, işletim sistemleri gibi klasik bilgisayar bilimleri dersleri bu programda yer almayabilir. Çünkü bölümün mezunları genellikle bu tür ürünlerin tasarımı ile uğraşmazlar. Onların uğraş alanları teknolojinin hızla ilerlemesine paralel olarak insanların giderek artan ihtiyaçlarını karşılayacak yeni uygulamalar için yazılımlar geliştirmektir. Bu yeni uygulamalar aynı zamanda geleneksel mühendislik teknolojilerinin yerine geçmekte ya da onları tamamlamaktadır. Oysa bilgisayar bilimlerinde genellikle çekirdek yazılım alanlarına odaklanılır. Bilgisayar bilimi bilgisayar yazılım ve donanımının tasarımı ve üzerinde gerçekleştirilen işlemler şeklinde tanımlanabilir. Bilgisayar bilimi kısaca bilgisayar teknolojisinin fen bilimlerine, iş dünyasına, ticarete, iş ve sanata uygulamasıdır.

Ülkemizde bilgisayar mühendisliği eğitimi olarak bilgisayar bilimleri eğitimi veriliyor olmasına rağmen, bilgisayar mühendisliği aslında elektronik mühendisliği ile bilgisayar bilimlerinin birleşimi olan bir disiplindir. Bilgisayar mühendisliği bölümünün eğitimi genellikle elektronik mühendisliği, yazılım tasarımı ile donanım tasarımının entegrasyonu şeklindedir.

Sonuç olarak, yazılım mühendisliğinin bir bilgisayar bilimleri programı olarak değerlendirilmesi mümkün değildir. Yazılım mühendisliği tüm mühendislik alanlarını içeren kümenin bir elemanı olarak kabul edilmesine rağmen, önceki bölümde de ifade edildiği gibi bu kümenin diğer elemanlarına benzer şekilde imal edilemez; sadece tasarlanabilir. Diğer mühendislik ürünlerinin zamanla aşınmasına rağmen,

yazılım aşınmaz; fakat eskir, yani kullanım potansiyeli düşer.

4. Formal Metotların Yazılım Mühendisliği Eğitimindeki Rolü

Bilgisayar bilimlerinde formal metotlar, yazılım ve donanım sistemlerinin geliştirilmesi sürecinde matematik temelli tekniklerden yararlanılması şeklinde tanımlanır. Bu da sistemin istenilen koşulları sağlayıp sağlamadığının saptanması, bir başka ifade ile sistemin doğrulanması ve özelliklerinin belirtilmesi demektir. Benzeri tanım gerçek dünya problemlerinin çözüldüğü ileri teknoloji kullanan günümüz yazılım mühendisliği uygulamaları için de yapılabilir. Geçmişte yazılım uygulamalarında formal metotların kullanımı fazla tercih edilmemekteydi; nedeni ise mevcut gösterimlerin ve tekniklerin açık olmaması, desteklenen araçların uygunsuzluğu ve kullanımlarının oldukça zor olması idi. Ayrıca işyerlerinde bu yöntemleri kullanacak düzeyde eğitimli insan sayısı da fazla değildi. Fakat 90'lı yıllarda endüstriyel yazılım tasarımı önem kazandıkça, geliştirilen sistemin özelliklerinin ayrıntılı dokümantasyonu için formal gösterimlerin önemi fark edilmiştir. Yazılımın belirtimi, yani spesifikasyonu ve donanımın sağlanması için formal metotlardan yararlanmak zorunlu hale gelmiştir.

Yazılımın belirtimi sözdizimi ve semantiği matematiksel olarak tanımlanmış bir dil kullanır. Sistemin özellikleri fonksiyonel davranış, zamanlama davranışı, başarım karakteristikleri ya da içyapı gibi farklı şekillerde tanımlanabilir. Zamanla formal belirtim davranışsal özelliklerin betimlenmesinde oldukça başarılı olmuştur. Farklı bir sistem durumu oluşturmak üzere, farklı belirtim dillerinin entegrasyonu ise çok tercih edilen bir yoldur. Benzer şekilde başarım, gerçek-zaman sınırlayıcıları, güvenlik politikaları ve mimarı tasarım gibi sistemin davranışsal olmayan durumları da formal betimleme özellikleri olarak alınabilir.

Diğer bir formal metot olan donanımın doğrulanması ise, model denetimi ve teorem kanıtı gibi tekniklerin endüstriye uyarlanarak şartlara daha uygun bir benzetimin gerçekleştirilmesidir. Model denetimi ile sistemin sonlu bir modeli oluşturularak modelin özelliklerinin kontrolü gerçekleştirilir. Denetim, model sonlu olduğu için kesin olarak sınırlanmış tam kapsamlı bir durum-uzay aramadır. Model denetiminin ilk uygulamaları genellikle donanım ve protokol doğrulanması idi. Teorem kanıtı ise, sistem ve sistemin özelliklerinin matematiksel mantıkla formüle edilmesidir. Formal sistem olarak adlandırılan bu mantık bir dizi aksiyom ve çıkarım kurallarından oluşur. Teorem kanıtı sistemin aksiyomlarından özelliğin ispatının bulunması işlemidir; ispatlar elle yapılabildiği gibi, makine destekli olarak da gerçekleşebilir.

Yazılım mühendisliği uygulamalarında yazılımın belirtimi ve donanımın sağlanmasında formal metotlardan yararlanılmasıyla, hem araştırmacılar hem de uygulayıcılar tarafından oldukça verimli endüstri ölçekli yazılım ürünleri geliştirilmiştir. Formal belirtim 90'lı yıllarda öncelikli olarak ticari ve güvenlik açısından kritik sistemler üzerinde uygulanmıştır.

4.1 Temel Formal Metotlar

Mühendisler problemlerin çözümüne yararcı, yani pragmatik yaklaşırlar. Herhangi bir yardımcı hesaplama aracı kullandıklarında problemi daha iyi tanımladıklarına ve böylece sonuçları daha etkin değerlendireceklerine inanırlar. Oysa matematikçiler için böylesi bir aracın kullanımı yararlı olmasına rağmen, yeterli değildir. Pek çok mühendislik disiplini uygulamasında matematik kullanımı ön planda olmasına rağmen, formal metotlara fazla önem verilmez. Ayrıca formal metotların yazılım geliştirmedeki öneminin, mühendislikte matematiğin etkisi ile eşdeğer olduğu söylenebilir.

Endüstriyel yazılım mühendisliğinde formal metotlardan yararlanmanın oldukça tartışmalı olmasının nedeni, yazılım maliyetini çok yükseltmesidir. Bu nedenle formal yöntemler, güvenlik ve gizliliğin önemli olduğu yüksek düzeyde yazılım tamlığı (bozulmamışlık) içeren sistemlerinin geliştirilmesinde daha yaygın kullanılır. Yazılım problemlerinin çözümünde formal metotlardan yararlanmanın temel amacı bilindiği gibi, önemli belirtim hatalarının doğuracağı riskleri azaltmaktır. Bu da günümüzde yalın (lightweight) formal metotların kullanılması ile gerçekleşir [3]. Yalın yaklaşımla, uygulamaya odaklanılmış herhangi bir formal metot kısmi olarak kullanıldığında, maliyet düşmekte ve uygulamadan daha fazla yarar sağlanmaktadır. Yalın yaklaşımın elemanları dilde, modellemede, analizde, birleşimde uygulamanın gerektirdiği kadar metodu kısmi olarak kullanırlar. Formal metotlara yalın yaklaşan örneklerden biri Alloy nesne modelleme sistemidir [4]. Bu sistem örnek olay kullanım (use

case) sürücülü geliştirme ile Z işaretleme sisteminin [5] bazı durumlarının sentezini gerçekleştirir. Alloy nesne modellemede kullanılan bir başka yalın örnek olarak, yine Z sistemine benzer belirtimleri olan "Vienna Development Methods" (VDM) [6] araçları verilebilir. Bir yazılım çalışmasında geliştirilen modelin özellikleri Z ve VDM gibi dillerin bazı aksiyomlarını, teorem çıkarım kurallarını ve ispat tekniklerini kullanarak analiz edilir. Bu tür sistemlerin en büyük yararı her bilgi düzeyindeki kullanıcının yararlanabilmesidir. Geliştirme aracı, basit bir model denetim programı ya da kişinin derin matematik bilgisine sahip olmasını gerektiren bir teorem çözücü de olabilir.

Problem çözümünde formal metotların başlıca yararı, geliştirilen ürünlerdeki belirsizlik ve kesin olmama durumlarının kaldırılarak, tümüyle nitelikli ürünler elde edilmesidir. Ayrıca analiz işlemlerinin otomatik olarak gerçekleşmesi istenilen özelliklerin görüntülenmesini, istenmeyenlerin ise kaldırılmasını sağlar. Örneğin hava trafik kontrolü gibi hatanın hiçbir şekilde toleransının mümkün olmadığı yüksek nitelikli ürünlerin geliştirilmesinde formal metotların kullanımı önemlidir. Bazı yöntemlerdeki soyutlama gücü ve otomasyon etkisi gittikçe karmaşıklaşan yazılımlara karşı önemli bir silahtır. Model-Sürücülü yazılım geliştirmede, formal yazılım yöntemleri ve gereksinimler tarafından desteklenen herhangi bir geliştirme metodolojisi, OMG ve IBM tarafından gerçekleştirilen uygulamalardaki gibi, birincil yapay olguları, yani bozulma etkisini kodun otomatik yaratılmasından oluşturur.

5. Yazılım Mühendisliği Eğitim Programları Çalışmaları

Yazılım mühendisliği programlarının düzenlenmesi ile ilgili ilk çalışmalar 1987 yılında "The Conference on Software Engineering Education and Training-CSEET" isimli konferansla başlamıştır. SEI (Software Engineering Institute) tarafından gerçekleştirilen bu eğitim konferansları tamamlanmış olmasına rağmen, toplantılara yıllık periyotlarla hala devam edilmektedir.

Diğer taraftan bir başka eğitim çalışması "The Quide to Software Engineering Body of Knowledge – SWEBOK" 1998 yılında "The Software Engineering Coordination Committee - SWECC" tarafından başlatılan bir projedir. Bu çalışmada öncelikle IEEE tarafından desteklenen yazılım mühendisliği standartları üzerinde odaklanılmıştır. SWEBOK aynı zamanda yazılım mühendisliği disiplininin sınırlarının ve özelliklerinin geniş katılımlı bir konsensüsle belirlenmiş olduğu ve yazılım mühendisliği disiplini destekleyen "Body of Knowledge" kavramına erişimin sağlanmasını hedefleyen bir projedir [7]. Eğitim programlarının sürekli olarak güncellenmesinde halen devam etmekte olan bu tür toplantıların rolü büyüktür. Ayrıca yazılım mühendisliği eğitiminde akreditasyon ölçütleri 1999 yılında geliştirilmiş ve 2003 yılında ilk programlar akredite olmaya başlamıştır.

5.1.Beykent Üniversitesi Yazılım Mühendisliği Programı

Gerçek şudur ki yazılım mühendisliği lisans programları ile ilgili ortak bir fikir birliği bulunmamaktadır. Beykent Üniversitesi'nin yazılım mühendisliği programı bu alanda konunun uzmanlarının hazırlamış olduğu " The Undergraduate SE Curriculum Model- SE2004" [8] çalışmasından Türkiye koşullarına uygun olanı seçilerek düzenlenmiştir. Buna göre 8 yarıyıllık eğitim süresince okutulacak dersler 4 ana başlıkta gruplanmaktadır:

- *Bilgisayar Bilimleri Dersleri*
  - *Programlama Dilleri (2 yarıyıl)*
  - *Java Programlama*
  - *Veri Yapıları ve Algoritmalar*
  - *Veri Tabanları Yönetim Sistemi*
  - *İşletim Sistemleri*
  - *Bilgisayar Organizasyonu ve Mimarileri*
  - *Bilgisayar Haberleşme ve Ağı*
- *Matematik Temel Dersleri*
  - *Ayrık Yapılar (2 yarıyıl)*
  - *Olasılık ve İstatistik*
  - *Algoritma Tasarımı ve Analizi*
- *Teknik Olmayan Mecburi Dersler*
  - *Mühendislik Ekonomisi*
  - *Yazılım Mühendisliği Uygulaması*
- *Yazılım Mühendisliği Temel Dersleri*
  - *Yazılım Mühendisliğine Giriş*
  - *Yazılım Mühendisliğinin Temelleri*
  - *Yazılım Geliştirme*
  - *Yazılım Mühendisliğinde İnsan Bilgisayar Etkileşimi*
  - *Yazılım Kalite Güvencesi ve Testi*
  - *Yazılım Tasarımı ve Mimarisi*
  - *Yazılım Gereksinmeleri ve Analizi*
  - *Yazılım Metrikleri*

- *Yazılım Projesi Yönetimi*
- *Mühendislik Projesi*
- *Bilgi Güvenliği*
- *Yazılım Mühendisliği Bitirme Projesi*

Takım çalışması yapabilme becerisinin kazanılması yazılım mühendisliği öğrencileri için öncelikli zorunluluktur. Bu nedenle öğrencilerin temel eğitimlerini tamamlandıktan sonra hazırlayacakları proje çalışmalarında, bu becerinin mutlaka elde edilmesi gerekir. Ayrıca bu programların en önemli farklılığı eğitimin endüstri ile iletişim içinde sürdürülmesi gayretleridir. Bunun da "Mühendislik Projesi" dersi ile gerçekleştirilmesi amaçlanmaktadır. Son yarıyıl hazırlanan bitirme projesi çalışması pek çok programda "Software Engineering Capstone Project" olarak farklı bir ana başlıkta sınıflandırılmakta; böylece bu çalışmanın önemi açık olarak vurgulanmaktadır.

6. Yazılım Mühendisliği Eğitiminin Geleceği

Yazılım mühendisliği eğitiminin ilerideki sorunları düşünüldüğünde, günümüz eğitim programları hazırlanırken göz önüne alınması gerekenler şöyle özetlenebilir [9]:

- Programların öğrenciler için cazip olacak şekilde hazırlanması,
- En etkili şekilde eğitime odaklanılması,
- Endüstri ile iletişimin daha aktif gerçekleştirilmesi,
- İleriye yönelik öğretim programlarının tanımlanması,
- Eğitimin mevcut öğrencilerin koşullarına göre gerçekleştirilmesi,
- Eğitime daha çok gösterim odaklı bir yapı kazandırılması,
- Yazılım mühendisliği eğitimi için temel altyapı gerektiğinin kabul edilmesi,
- Yazılım mühendisliği eğitim araştırmalarının nitelik ve saygınlığının arttırılması.

Yukarıda verilenlerin tümünü, hatta birkaçını sağlamak bile hesaplama disiplini açısından çok olumludur. Özellikle son üç madde sadece bilgisayar bilimlerini değil, aynı zamanda yeni bir disiplin olan bilgi teknolojilerini de kapsamaktadır.

90'lı yılların ikinci yarısından itibaren eğitim programlarını hazırlayanlar, olgunluk düzeyinde bir meslek edinimi için temel taşlar olan profesyonel eğitimin başlangıcı, akreditasyon, becerilerin geliştirilmesi, sertifikasyon, lisans alma, profesyonel gelişim, kodların etiği, profesyonel toplum gibi faktörleri göz önüne alarak çalışmışlardır [10]. Fakat günümüze bakıldığında tam olarak olgunlaşmış bir meslek düzeyine erişilemediği, programların temel yapı taşlarının pek çoğunun sağlanmadığı görülür. Bu nedenle de şimdiye kadar görülen başarılarla yetinmeyip, bu yeni disiplini ilerletmeye devam edilmelidir. Programların teknik sınıflandırmasında fazla önemi olmamasına rağmen, ekip çalışması ve iletişime önem veren ve endüstri ile bütünleşmiş bir yazılım mühendisliği eğitim programı günümüz öğrencilerini daha fazla mutlu edebilir.
Sonuç olarak; bir yazılım mühendisliği programı aşağıdaki beş tamamlayıcı elemanı mutlaka içermelidir:

- *İlkeler:* tüm çalışma alanı için geçerli olan kalıcı kavramlar;
- *Alışkanlıklar:* profesyonellerin bilinçli ve düzenli olarak uyguladıkları problem çözme teknikleri;
- *Uygulamalar:* ilkelerin ve alışkanlıkların en iyi şekilde ifade edildiği uzmanlık alanları;
- *Araçlar:* İlke ve alışkanlıkların uygulanmasını kolaylaştıran çağdaş ürünler;
- *Matematik:* Yapılanların tümünün anlaşılmasını sağlayan formal temeller.

SWEBOK 2010 çalışma takımı Software Engineering 2004 programlarını güncelleme çalışmalarına devam etmektedir. Programlardaki en önemli yeniliğin güvenlik olması çok doğaldır. Zira amatör korsanlar, ticari rakipler, kişisel suçlular, küçük suçlu grupları, içeriden saldıranlar, organize suç konsorsiyumu, psikopat ve sosyopatlar, sosyal protestocular ve teröristler potansiyel hücum sahiplerinden bazılarıdır.

8. Kaynaklar

[1] www.aip.org/statistics American Institute of Physics, The Research Center.
[2] Parnas D.L., "Software Engineering Programs Are Not Computer Science Programs," *IEEE Software*, vol. 16, no. 6, pp. 19-30, 1999..
[3] Jackson D. and Wing J., "Lightweight Formal Methods", IEEE Computer, 1996.

[4]Jackson D. , “Alloy: a Lightweight Object Modeling Notation”, ACM Transactions on Software Engineering and Methodology (TOSEM), 2002.
[5] Davies J. and Jim Woodcock J., “Using Z: Specification, Refinement and Proof “, Prentice Hall, 1996.
[6] Fitzgerald, J.S., Larsen, P.G., Mukherjee, P., Plat, N. and Verhoef, M., “Validated Designs for Object-oriented Systems”, Springer Verlag, 2005.
[7] Guide to Software Engineering Body of Knowledge 2004 SWEBOK, IEEE Press.
[8] Conputing Curricula, Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE Computer Society and Association of Computing Machinery.
[9] Hislop G.W., “Software Engineering Education: Past, Present and Future”, in Software Engineering Effective Teaching and Learning Approaches and Practices, Information Science, 2009.
[10] Ford G.A., Gibbs N., “A Mature Profession of Software Engineering”, SEI,Technical Report CMU/SEI-96-TR-04, 1996.