Bileşen Yönelimli Yazılım Ölçütleri

Nael SALMAN

Bilgisayar Mühendisliği Bölümü, Çankaya Üniversitesi

e-posta: nsalman@cankaya.edu.tr

Özet

Bu çalışma bileşene yönelik olarak geliştirilen yazılım sistemleri için karmaşıklık ölçütleri sunmaktadır. Karmaşıklık ölçütlerinin geçerlilik kriterleri tanımlanmış ve bu ölçütlere uygulanmıştır. Geçerleme çalışması için öğrenci projeleri kullanılarak bir deney yapıldı. İnceleme sonuçları, sözü geçen ölçütlerin sistem tasarımı ve bakımı ile ilgili çaba ve maliyet irdelemeleri için çok önemli olabileceğini gösterdi.

Abstract

A set of complexity metrics for component oriented software systems has been defined. The metrics have been also evaluated and validated. Validation process has been performed by conducting a case study using graduate students projects designed for component oriented software development. The case study results reveal the importance of the mentioned metrics in the investigation of effort and costs related with system design and maintenance.

1. Giriş

Soyutlama, yazılım geliştirme tarihi boyunca sistemlerin temel yapısını oluşturan birimleri belirlemek için kullanılagelen etkin bir metod olmuştur. İlk yıllarda fonksiyonlara dayalı soyutlama yapılıyordu [1]. Seksenli yıllarda nesneye yönelik (NY) yaklaşım kendisini göstermeye başladi. NY yaklaşımda veri ve fonksiyonlar 'sınıf' denilen bir tek birim içinde paketleniyor ve bir sınıf birden çok nesneyi tanımlayabiliyor. Sınıf soyutlaması kullanıldığında daha büyük ve daha güçlü sistemlerin daha ucuz bir şekilde geliştirilmesi mümkün oldu [2]. Günümüzde ise bileşen yönelimli sistem geliştirme yaygın olarak kullanılmaya başlandı.

Bileşen yönelimli yazılım mühendisliğinin (BYYM) yazılım geliştirmek için en yaygın ve en çok tercih edilen yaklaşım olacağını çok yakın zamanda göreceğiz [3]. Bileşen yönelimli yazılım mühendisliğinde büyük sistemlerin geliştirilmesi daha kolay ve bunun sadece bileşenleri bütünleştirerek yapılabilmesi mümkündür. Ayrıca bileşenler kullanılarak çoğu bakım problemi daha kolay bir şekilde çözülebilir [4, 5, 6]. Bileşen yönelimli sistem geliştirmeye olan ilgi sadece bilimsel çevrelerce sınırlı değil. Bir çok büyük yazılım şirketi bu konuya yatırım yapmaya başladı. Örnek olarak JavaBeans ve DCOM bileşen protokollerini gösterebiliriz; Bu protokoller sahanın en büyük organizasyonları tarafından ortaya konmuştur.

Bileşen yönelimli yazılım mühendisliğinin tam bir yaklaşım olabilmesi için sistemlerin geliştirme ve bakım maliyetlerinin hesaplanması, ayrıca güvenilirlik, verimlilik, bütünleşme karmaşıklığı, sınama masrafları, ve tasarım mimari kalitesinin ölçülebilmesi için ölçütlerin hazırlanması çok

önemlidir [5, 7, 9]. Biz bu çalışmada yazılım geliştirme süreçleri ve ürünlerinin en önemli özelliklerinin ölçülebilmesi amacıyla bir ölçüt kümesi hazırladık ve bu ölçütlerin geçerlilik testlerini yaptık.

Hazırlanan ölçütlerin tanımlanması ve ölçütlerin geçerleme metodları bu makalenin ilerleyen kısımlarında yer almaktadır. Bölüm 2'de, önerilen ölçütlerin tanımları içerilmektedir. Bölüm 3'te, önerilen ölçütlerin geçerlenmesi. Bölüm 4'te, elde edilen sonuçların açıklaması ve Bölüm 5'te çalışma ile ilgili sonuçlar bulunmaktadır.

2. BYYM Sistemlerinin Karmaşıklığı ve Karmaşıklık Ölçütleri

Birinci bölümde gösterildiği gibi ve BYYM ile ilgili yazın inceleme sonuçlarına göre BYYM'nin karmaşıklığının tanımlanmadığı ve BYYM'ye adanmış karmaşıklık ölçütlerinin bulunmadığı ortaya çıkmaktadır. Öte yandan, BYYM sistemlerinin karmaşıklık ölçütlerinin bulunması, hem yazılım geliştiricileri hem de yazılım projelerinin idarecileri ve takım liderleri açısından çok önem taşımaktadır. Bu yüzden, BYYM kullanılarak geliştirilen sistemlerin karmaşıklık çalışmalarını yaptık ve BYYM sistemlerinin karmaşıklığının üç farklı seviyede tanımlanabildiği çıkarımında bulunduk. Bölüm 2.1'de BYYM sistemlerinin karmaşıklığı tanımlanmakta ve farklı seviyeleri belirlenmektedir. 2.2-2.4 bölümlerinde ise 2.1 bölümünde tanımlanan karmaşıklığın farklı seviyelerine ilişkin ölçütler tanımlanmaktadır.

2.1 BYYM'nin Karmaşıklığı

Yazılım mühendisliği literatürüne incelendiğinde karmaşıklık terimin çok farklı şekillerde tanımlanıp uygulandığını görebiliriz. Bazı yazılım mühendisliği araştırmacılarına göre bir yazılım sisteminin karmaşıklığı o sistemin büyüklüğüyle doğrudan alakalıdır. Öte yandan genelde çoğunluğu temsil eden araştırmacılara göre sistemin karmaşıklığı, büyüklüğüne değil sistemin güvenilirliği (reliability), bakım kolaylığı (maintainability), ve anlaşılabilirliği (understandability) gibi sistem özelliklerine bağlıdır [9, 10]. Bileşen yönelimli sistemlerde en çok sistemlerin mimarisine önem verilmektedir. Bunun için bu çalışmada yer alan karmaşıklık yaklaşımımızda en çok birimler arasındaki ilişkilere ve bunların hiyerarşisine ağırlık veriyoruz.

Bileşen yönelimli bir sistem, arayüzleri tanımlanmış bileşenlerden oluşmaktadır. Bileşenler kendi aralarında konuşabilir ve birbirlerine servis sağlayabilirler. Bir bileşenin içine baktığımızda bileşenin temel yapı bloklarını metodların oluşturduğunu görürüz. Karmaşıklığını ölçmek istediğimiz bileşenin içindeki metodların bireysel karmaşıklıkları ve metodlar arasındaki ilişki ve iletişimlerden kaynaklanan karmaşıklıkları göz önüne almamız gerekmektedir. Metod ve metodlar arası ilişkiler dışında bir bileşenin dışarıya verebileceği servisler o bileşenin arayüzlerinden alınabilir. Arayüz ve arayüzların sayılarına göre bileşenlerin karmaşıklıkları artar ya da azalabilir. Onun için bir bileşenin karmaşıklığı onun arayüzleriyle doğrudan bağlantılı bir özellik olarak değerlendirilmektedir.

Bileşen yönelimli bir sistemi tasarlarken bileşenler, bileşenlerin bağlantıları, bileşenler arası ilişkiler, ve bileşenlerin hiyerarşilerini de tasarlamamız gerekir. Bir sistemin işlevlerini gerçekleştirebilmek için farklı tasarımlar oluşturabiliriz. Bu farklı tasarımlar arasından seçim yaparken dikkate aldığımız faktörlerin içinde hiyerarşi ve bileşenlerarası ilişkiler de bulunmaktadır.

Sonuç olarak yaklaşımımız sistemin karmaşıklığını üç farklı seviyede incelemektedir. Bu seviyeler ise, en altta metodların karmaşıklıkları, sonra bileşenlerin karmaşıklıkları, ve en üstte sistem karmaşıklığı olarak tanımlanmıştır. Bu üç seviyede bulunan karmaşıklıkları ölçmek için kolay bir şekilde hesaplanabilecek ve anlamlı bilgi taşıyan bir grup ölçüt önerilmeye çalışılmıştır.

2.2 Seviye 1: Metodların Karmaşıklığı Ve Ölçütleri

Bu bölümde bileşenlerin en alt seviyede görülebilenen karmaşıklıkları irdelenmektedir. Bu karmaşıklık bileşenin metodlarının karmaşıklıklarından kaynaklanmaktadır. Metod karmaşıklıkları aşağıda açıklanan ölçütler kullanılarak tanımlanabilir.

Ölçüt 1.1: Metodların Çevrimsel Karmaşıklığı. Çevrimsel karmaşıklık **(ÇK)** [11] ölçütü yıllardır etkili bir şekilde bir işlevin karmaşıklığını ölçmek için kullanılmaktadır. ÇK ölçütü kullanarak hesaplanan metod karmaşıklığı, o metoda ne kadar sınama zamanı harcanması gerektiğinin rahat bir şekilde tahmin edilmesinde yararlı olur.

Ölçüt 1.2: Diğer metodlara yapılan çağrı sayısı (DMÇS). Bu ölçütün değerini hesaplamak için bir metodun çağırdığı metod sayısı bulunur. Bu ölçütün değeri metodun diğer metodlara ne kadare bağımlı olduğunu gösterir. Bir metodun işini tamamlayabilmesi için diğer metodların sağladığı servislere ne derecede ihtiyaç duyduğu, o metodun bağımlılığını gösterir. Metodun anlaşılması için çağırdığı bütün metodları anlamamız gerekir ve bu da metodun karmaşıklığını artırmaktadır.

Bir metodun toplam karmaşıklığı hem kendi yapısından hem de bağımlı olduğu metodlardan kaynaklanmaktadır. Dolayısıyla metodun yapısal karmaşıklığı ile ilgili ÇK ölçütünü ve metodun bağımlılığını hesaplamak için DMÇS ölçütünü kullanacağız. Bir metodun (*m)* toplam karmaşıklığı MKAR(m), ÇK ve DMÇS'den oluşan bir fonksiyon *f* olarak görülebilir ve aşağıdaki denklemlerde gösterildiği gibi hesaplanabilir.

$$\text{MKAR (m)} = \text{f (ÇK(m), DMÇS(m))}$$

ÇK ve DMÇS değerleri yüksek olunca olumsuz, düşük olunca olumlu bir karmaşıklık anlamına gelir. Bu yüzden MKAR değerini, ÇK ve DMÇS'un eşit ağırlıklara sahip oldukları farzedilerek doğrudan toplanmaları şeklinde hesaplayabiliriz. Buna göre MKAR değeri denklem (1) deki gibi hesaplanır.

$$\text{MKAR(m)} = \text{ÇK(m)} + \text{DMÇS(m)} \quad (1)$$

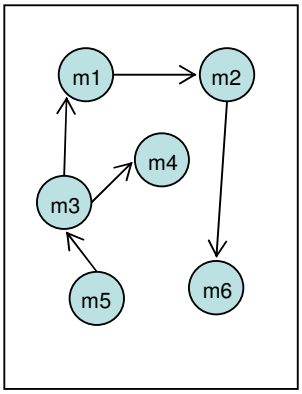
Bir bileşen, C, için toplam metod karmaşıklığı (TMKAR) denklem (2)'te gösterildiği gibi hesaplanabilir.

$$\text{TMKAR(C)} = \sum_{i} MKAR(m_i) \quad (2)$$

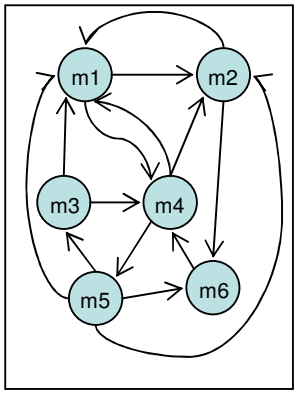
2.3 Seviye 2: Bileşenlerin Karmaşıklığı ve Ölçütleri

Karmaşıklık analizi yaplırken aşağıdan-yukarıya yaklaşımı izlendi. Bölüm 2.1'de en alt seviyedeki metodlara karşı düşen bir değerlendirme sunulmuştu. Bunun bir üst seviyesi ise sistemin temel yapısını oluşturan bileşenler ile ilgilidir. Bir bileşenin karmaşıklığının farklı faktörlerden kaynaklandığını görmekteyiz. Bu faktörlerin en önemlilerden birisi, bileşenin içindeki metodlar arasındaki ilişkilerdir. Bu bileşenin iç mimari açısından ne kadar "karmaşık" olduğunu göstermektedir. İkinci faktör ise bileşenin dışarıya ne kadar bağımlı olduğu ile alakalıdır. Bileşenin dışarıya olan bağımlılığını ölçmek için o bileşenin diğer bileşenlerle bağlantılılık (coupling) sayısına bakabiliriz. Üçüncü faktör ise bir bileşenin sahip olduğu arayüz sayısıdır. Bu üç faktörden kaynaklanan bileşen karmaşıklığını değerlendirmek için kullandığımız ölçütler ve bu ölçütlerin kullanımına dair beklentilerimizi aşağıdaki gibi açıklayabiliriz.

Ölçüt 2.1: İç metodlar arası çağrı sayısı (İMÇS): Bu ölçüt, metodların birbirine yaptığı çağrıları sayarak hesaplanabilir. Beyaz kutu yaklaşımı ile bir bileşeni grafiksel olarak temsil edersek, metodlar grafikteki düğümlere karşı düşmektedir. Şekil 1.a ve şekil 1.b'ye baktığımızda eğer iki şekildeki verilen görevler aynıysa ve diğer etmenler sabit varsayılırsa, şekil 1.a'daki tasarımın daha iyi olduğu kolayca söylenebilir.



Şekil 1.a. Az bağlantılı model



Şekil 1.b. Çok bağlantılı model

Ölçüt 2.2: Bileşenin arayüz sayısı (BAYS): Bu ölçütü hesaplamak için sadece bileşenin arayüzleri sayılır. Bu ölçütün amacı, bileşenler arasındaki ilişkilerden kaynaklanan karmaşıklığı hesaplamaktadır. Temel düşüncemize göre:

- Arayüz sayısının çok olması durumunda, bileşenler arasında çok ilişki bulunuyor ve bu bileşenlerin birbirine çok bağımlı olduğu anlamına gelebilir. Bileşenler birbirine çok bağımlı oldukları zaman onların test, birleştirme ve bakım işlemleri daha zor olacaktır.
- Arayüz sayısı yüksek olması arayüzleri içeren bileşenin çok hizmet verdiği anlamına gelebilir. Çok hizmet veren bir bileşen, fazla özelleşmiş bir bileşen anlamına gelebilir. Buna göre de farklı sistemlerde tekrar kullanılabilme ihtimali düşüktür.

Ölçüt 2.3: Diğer bileşenlerle ilişki sayısı (DBİS): Bu ölçütü, bileşenin kaç farklı bileşenden servis istediğini saymakla elde edebiliriz. Bir bileşen başka bir bileşenden servis isterse ona bağımlı olduğu anlamı çıkar. Bileşenler arası bağımlılık, istenmeyen bir tasarım özelliğidir

çünkü bu sonuç kötü ayrıştırmadan kaynaklanır. DBİS değerinin sistem karmaşıklığına olan etkisini şöyle özetleyebiliriz:

- Bir bileşenin DBİS değerinin yüksek olması o bileşenin diğer bileşenlere çok bağımlı olduğu anlamına gelir. Bağımlılık, anlama, bakım, ve birleştirme zorluklarına neden olur. Bu yüzden bileşenler arasındaki ilişkilerin olabildiğince aza indirilmesi gerektiğini düşünüyoruz.
- Bir bileşen çok bağımlı olduğu zaman onu yeniden kullanma ihtimali çok azdır.
- Çok bağımlı bileşenlerden oluşan bir sistemin geliştirilmesi daha zor ve bakımı daha masraflıdır.

Yukarıda bahsi geçen etkenlere baktığımızda, bir bileşenin, C, karmaşıklığı bu üç etkenden oluşan bir fonksiyon, *f,* olarak belirtilebilir:

$$\text{BKAR}(C) = f(\text{İMÇS}(C), \text{BAYS}(C), \text{DBİS}(C))$$

İMÇS, BAYS ve DBİS ölçütlerinin birer pozitif sayı ürettiğini görmekteyiz. Üçü için de düşük değer istenmektedir. Bileşen karmaşıklığını, azaldıkça iyi olarak yorumlanacak tek bir büyüklük ile ifade etmek istiyoruz. BKAR değeri denklem (3)'te görüldüğü gibi belirtebiliriz:

$$\text{BKAR}(C) = \text{İMÇS}(C) + \text{BAYS}(C) + \text{DBİS}(C) \quad (3)$$

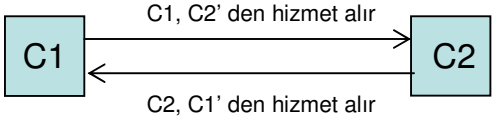
Bileşen yönelikli bir sistem, S, için tüm bileşenlerinin bireysel karmaşıklıkları toplamı (TBKAR) ise denklem (4)'te görüldüğü gibi belirlenebilir:

$$\text{TBKAR}(S) = \sum_j \text{BKAR}(C_j) \quad (4)$$

2.4 Seviye 3: Sistem Karmaşıklığı ve Ölçütleri

Bileşenleri birleştirdiğimiz zaman yeni bir karmaşıklık doğabilir (GSKAR). Bu yeni karmaşıklık, yeni doğabilecek bileşenlerarası ilişkilerden, ya da bileşenlerin hiyerarşisi yüzünden doğan bir karmaşıklıktır. Bu karmaşıklığı ölçmek için sistem mimarisine dayalı iki yeni ölçütü şöyle tanımlıyoruz:

Ölçüt 3.1: Bileşenlerarası bağlantı sayısı (BABS). Bu ölçütün hesaplanması bileşenlerarası bağlantıları sayarak yapılır. Bir sistemin grafiksel modelinden (örneğin COSEML [5] modelinden) doğrudan elde edilebilir. Bir bileşenin başka bir bileşenden servis alması bu iki bileşen arasında bağlantı var anlamına gelmektedir. Şekil 2'de C1 ve C2 bileşenleri karşılıklı olarak birbirlerinden servis almaktadır. BABS değeri hesaplanırken bu iki bağlantı ayrı olarak hesaplanır.



Şekil 2. Bileşenlerarası ilişkiler

Bileşenler arası ilişkilerin sistem karmaşıklığına etkisini şöyle özetleyebiliriz:

- Çok ilişkinin var olması, yüksek bağımlılık anlamına gelmektedir. Yüksek bağımlılık yüksek test ve bakım masraflarına neden olabilir.
- Bileşenlerarası çok bağlantı olması, çok mesaj akışına neden olabilir. Bu da ağ sorunu yaratır ve performansı etkileyebilir.

Ölçüt 3.2: Birleştirme hiyerarşisinin derinliği (BHİD): Bu ölçütün değerini sistem modelindeki seviyeleri sayarak elde edebiliriz. BHİD ölçütünün değerine göre sistemden bazı beklentilerin tahmininde bulunabiliriz. Bu tahminler şöyle özetlenebilir:

- Birleştirme hiyerarşisinin derin olması bileşen ayrıştırmasının iyi yapıldığı anlamına gelir ve sistemdeki bileşenlerin yeniden kullanılabilme ihtimali yüksektir.
- Öte yandan birleştirme hiyerarşisinin derinliği, sistemin birleştirilmesine harcanacak zamanın yüksek olacağına işaret edebilir ve bu yüzden maliyet artabilir.

Seviye 3'te sistem karmaşıklığı bir bileşen yönelimli sistem S için bir fonksiyon *f* olarak tanımlanabilir ve şöyle belirtilebilir:

$$GSKAR(S) = f\,(BABS(S), BHİD(S))$$

Sistemin toplam karmaşıklığı ise yukarıda bahsi geçen ölçütlerden oluşan bir fonksiyon *f* olarak şöyle belirtilebilir:

$$TSKAR(S) = f(\sum_{j} TMKAR(C_j)\,, TBKAR(S), GSKAR(S))$$

TMKAR(C), TBKAR(S), GSKAR(S) ölçütlerinin tümünün birer pozitif sayı ürettiğini görmekteyiz. Üçünde de yüksek değer istenmez; düşük değer istenir. Karmaşıklığı, değerinin düşük olmasının olumlu yorumlanacağı tek bir sayı olarak denklem (5)'te görüldüğü gibi belirtebiliriz:

$$TSKAR(S) = \sum_{j} TMKAR(C_j) + TBKAR(S) + GSKAR(S) \qquad (5)$$

3. Ölçütlerin Değerlendirilmesi

Ölçütlerin geçerliliğini sınamak için bir değerlendirme çalışması yapıldı. Bu çalışma için ODTÜ Bilgisayar Mühendisliği bölümünde geliştirilmiş yirmi farklı yüksek lisans öğrenci projesi incelendi. Projelerin tümü BYYM yaklaşımı ile COSEML [4] kullanılarak tasarlanmıştır. Yukarıda bahsi geçen ölçütlerin çoğu proje tasarımlarından toplandı ve ölçütlerin değerleri, geliştirici verimliliği, tasarım zamanı ve bakım zamanı ile ilişkileri açılarından incelendi. İncelemenin sonuçları şöyle özetlenebilir:

1. Yüksek tasarım zamanı ile yüksek DBİS değeri arasında doğrudan ilişki gözlemlendi.
2. Yüksek bakım zamanı ile ölçütlerin hepsinin yüksek değerleri arasında doğrudan bir ilişki görüldü.
3. Geliştirici verimliliğinin ölçüt değerleri yükseldikçe düştüğü gözlendi. En çok etki gösteren BHİD ve BAYS ölçütleri oldu.
4. İnceleme sonuçları sistemin güvenilirliği ile ilgili hiç bir bilgi vermedi. Bunun sebebi de bu inceleme yapıldığı zaman sadece projelerin tasarımlarının hazır olmasıydı. Sistem

güvenilirliğinin değerlendirilebilmesi için, uygulaması bitirilmiş projelerin bulunması gerekmektedir.

5. Sonuçlar konuya devam etmek için özendirici olarak yorumlanmaktadır.

Bu kısa sonuç değerlendirmesinde ölçütlerin beklentilere karşı düştükleri görünüyor. Doğal olarak daha kesin yargılara varabilmek için endüstride bitirilmiş projeler üzerinde çalışmaların yapılması gerekmektedir.

4. Sonuç

Çalışmada BYYM karmaşıklıkları ve karmaşıklık ölçütleri tanımlandı. Ölçütlerin gerçek hayata etkisi incelendi ve iyi sonuçlar alındı. Yapılan değerlendimenin daha etkin ve kesin olabilmesi sadece tasarımlar ile değil tamamen uygulanmış projeler ile çalışmanın genişletilmesine bağlıdır. Çalışmanın devamını BYYM yaklaşımına göre tasarlanıp uygulanacak projeler üzerinde sürdürmeyi planlamaktayız.

Geliştiricilerin büyük bir kısmı ölçütlerin tasarımlardan toplanması çabasını sıkıcı ve zaman alıcı bir iş olarak yorumlamaktadır. Dolayısıyla, çalışmanın devamı olarak bir otomatik ölçüt toplama aracı geliştirmenin önemi açıktır. Bu otomatik araç, ölçütleri tasarımlardan toplayacağı için bir tasarım aracı ile tümleştirilmelidir.

Kaynakça

[1]. Watt, D. A., Findlay, W., Hughes, J., "Programming Language Concepts and Paradigms," Prentice Hall, s. 1-20, 1990.

[2]. Sebesta, R. W., "*Concepts of Programming Languages*," Fifth Edition, Addison Wesley, s. 458-477, 2002.

[3]. Ravichandran, T., ve Rothenberger, M., "Software Reuse Strategies and Component Markets," Communications of the ACM, vol. 46, No. 8, s. 109-114, 2003.

[4]. Dogru, A. H., Tanik, M., "A process Model for Component Oriented Software Engineering," *IEEE Software*, March/April, , s. 34-41, 2003.

[5]. Vitharana, P., Zahedi, F. M., Jain, H., "Design Retrieval and Assembly in Component-Based Software Development," Communications of the ACM, vol. 46, No. 11, s. 97-102, 2003.

[6]. Basili, V. R., Boehm, B., "COTS-Based Systems Top 10 List," *IEEE Computer*, vol. 34, No. 5, s. 91-93, 2001.

[7]. Sedigh-Ali, S., Ghafoor, A., Paul, R. A., "Software Engineering Metrics for COTS-Based Systems," *IEEE Computer*, vol. 34, No. 6, s. 44-50, 2001.

[8]. Visaggio, G., 1997, "Structural Information as a Quality Metric in Software Systems Organization," *Proceeding of ICSM*, pp. 92-99.

[9]. Tian, J., Zelkowitz, M. V., "Complexity Measure Evaluation ve Selection," *IEEE Transactions on Software Engineering*, vol. 21, No. 8, s. 641-650, 1995.

[10]. Zuse, H., "Criteria for Program Comprehension Derived from Software Complexity Metrics," Proceedings of the Second International Workshop on Software Comprehension, IEEE, Capri/Italy, s. 8-16, 1993.

[11]. McCabe, T. J., "A complexity Measure," *IEEE Transactions on Software Engineering*, Vol. 2, No. 4, s. 308-320, 1976.