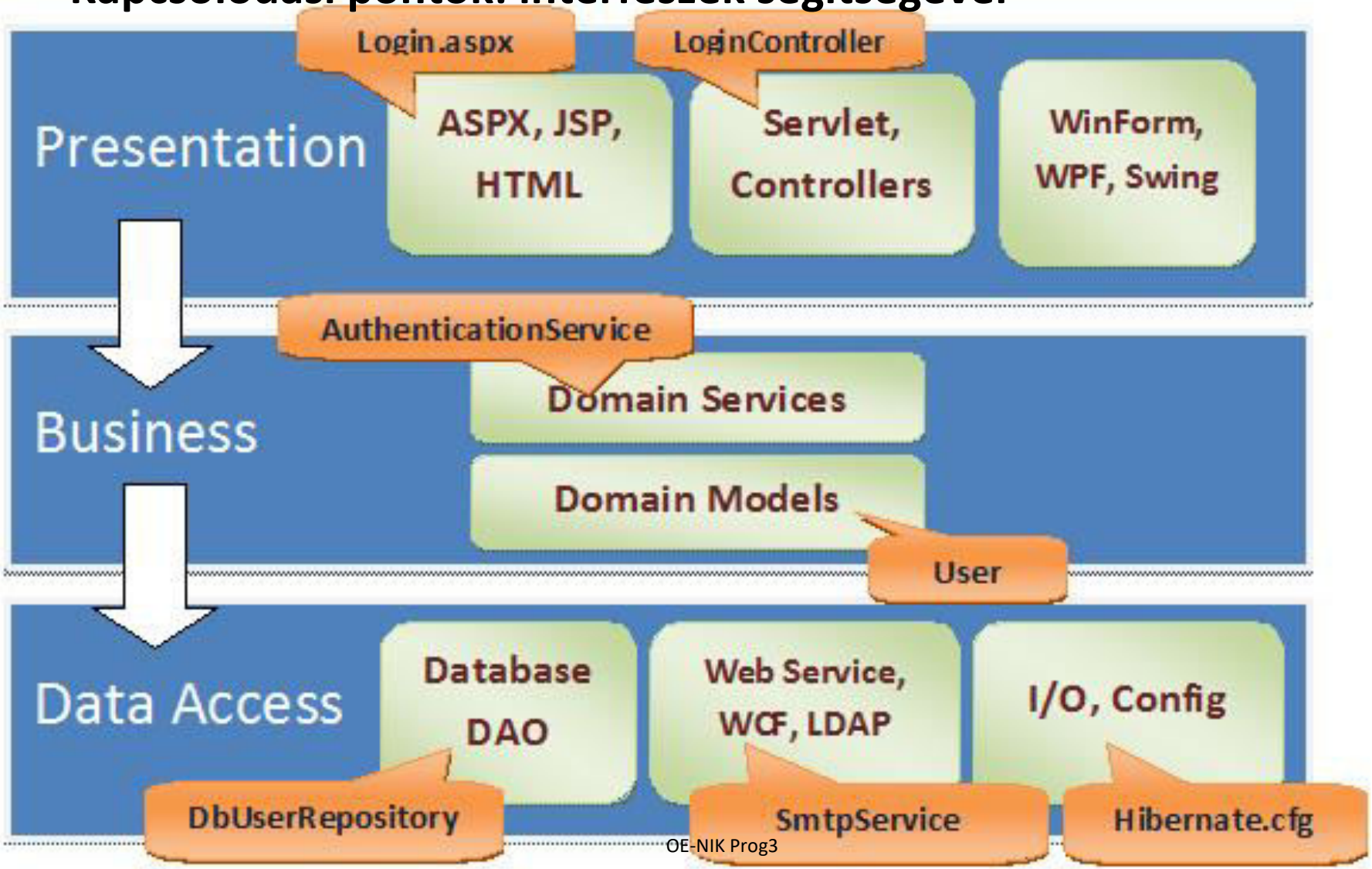
Haladó fejlesztési technikák

Projektmunka / Féléves Feladat

Tipikus Szoftver rétegek

- **Mindegyik réteg felbontható osztályokra/rétegekre (SOLID!)**
- **Kapcsolódási pontok: interfészek segítségével**



Használandó eszközök és struktúra

- **Projektfeladat: a weboldalról letölthető a roadmap + elvárások**
  - Szabályok: a /prog3 weboldalról letölthető prog_tools és prog3_requirements dokumentumok
- **Használandó eszközök (ld. később)**
  - Egy felhasználós, egy branch-et használó GIT repository
  - A kód legyen FXCop/StyleCop-helyes, **NULLA** warning/build error
  - DoxyGen segítségével generált HTML/CHM formátumú fejlesztői dokumentáció (PDF: opcionális, LaTeX segítségével)
- **Rétegzett projekt struktúra, min. 5 dotnet core C# projekt**
  - **Data**: Adatbázis + Entity Framework az eléréshez
  - **Repository**: EF adatkezelő metódusok (IQueryable eredményekkel)
  - **Logic**: Egy vagy több repository metódust felhasználó BL metódusok (CRUD + Non-Crud, utóbbi tipikusan LINQ lekérdezésekkel, lista eredményekkel)
  - **Test**: Unit tesztekkel ellátott kód (az üzleti logika osztályaira, mockolt Crud és mockolt/assertezett Non-Crud tesztekkel)
  - **Program**: Menüvezérelt (ConsoleMenu-simple, EasyConsole) konzol alkalmazás, példányosítások Factory segítségével

Rétegek: DATA (+ MDF, LDF via .gitignore!)

```
public partial class CarDbContext : DbContext
{
    public virtual DbSet<Brand> Brand { get; set; }
    public virtual DbSet<Car> Cars { get; set; }

    public CarDbContext()
    {
        this.Database.EnsureCreated();
    }
```

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.
            UseLazyLoadingProxies().
            UseSqlServer(@"data source=(LocalDB)\MSSQLLocalDB;attachdbfilename=|
              DataDirectory|\CarDb.mdf;integrated
              security=True;MultipleActiveResultSets=True");
    }
}
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
```

Rétegek: REPOSITORY

```
public interface IRepository<T> where T : class
```

```
public interface ICarRepository : IRepository<Car>
```

```
public interface IBrandRepository : IRepository<Brand>
```

```
public abstract class Repository<T> : IRepository<T> where T : class
{
    protected DbContext ctx;
    public Repository(DbContext ctx)
    {
        this.ctx = ctx;
    }
    public IQueryable<T> GetAll()
    {
        return ctx.Set<T>(); // "Set" as a noun, not as a verb!!!
    }
```

```
public class CarRepository : Repository<Car>, ICarRepository
{
    public CarRepository(DbContext ctx) : base(ctx) { }
```

```
public class BrandRepository : Repository<Brand>, IBrandRepository
```

Rétegek: REPOSITORY

```
public abstract class EfRepository<TEntity> :
    IRepository<TEntity> where TEntity : class
{
    DbContext context;
    // context.Set<TEntity>() => the table that stores TEntity

    public void AddNew(TEntity newInstance)
        { /* generic implementation */ }
    public void DeleteOld(TEntity oldInstance)
        { /* generic implementation */ }
    public IQueryable<TEntity> GetAll()
        { /* generic implementation */ return null; }

    public abstract TEntity GetById(int id);
}
```

Rétegek: LOGIC – SOLID

```
// Should add DTO instead of Entities => SKIP for this semester
// This is a SERIOUS security hole!
// var car = logic.GetOneCar(40);
// car.Model = "NEW NAME"; // Shouldn't be able to do this.
// logic.ChangeCarPrice(40, car.car_baseprice); // saves new model name too!!!
public class AveragesResult
{
    public string BrandName { get; set; }
    public double AveragePrice { get; set; }
    public override string ToString()...
    public override bool Equals(object obj)...
    public override int GetHashCode()...
}
public interface ICarLogic
// Avoid: GOD OBJECT, too many responsibilities!
// SPLIT UP into multiple classes AS YOU WANT!
{
    Car GetOneCar(int id);
    void ChangeCarPrice(int id, int newprice);
    IList<Car> GetAllCars();
    IList<AveragesResult> GetBrandAverages();
}
public class CarLogic : ICarLogic
```

Rétegek: LOGIC – SOLID + DI + CRUD

```
// Avoid god object => split up Logic into multiple classes
ICarRepository carRepo;
IBrandRepository brandRepo;
public CarLogic(ICarRepository carRepo, IBrandRepository brandRepo)
{
    this.carRepo = carRepo;
    this.brandRepo = brandRepo;
}
```

```
public void ChangeCarPrice(int id, int newprice)
{
    carRepo.ChangePrice(id, newprice);
}
public ILi
{
    return
}
```

```
public int AddBrand(string brandName)
{
    return brandRepo.Add(brandName);
}
public IList<Brand> GetAllBrands()
{
    return brandRepo.GetAll().ToList();
}
public IList<Car> GetCarsByBrand(int brand)
{
    return carRepo.GetAll().Where(x => x.BrandId == brand).ToList();
}
```

Rétegek: LOGIC – NON-CRUD

```
public IList<AveragesResult> GetBrandAverages()
{
    var q = from car in carRepo.GetAll()
            group car by new { car.Brand.Id, car.Brand.Name } into grp
            select new AveragesResult()
            {
                BrandName = grp.Key.Name,
                AveragePrice = grp.Average(car => car.BasePrice) ?? 0
            };
    return q.ToList();
}
public IList<AveragesResult> GetBrandAveragesJoin()
{
    var q = from car in carRepo.GetAll()
            join brand in brandRepo.GetAll() on car.BrandId equals brand.Id
            let item = new { BrandName = brand.Name, Price = car.BasePrice }
            group item by item.BrandName into grp
            select new AveragesResult()
            {
                BrandName = grp.Key,
                AveragePrice = grp.Average(item => item.Price) ?? 0
            };
    return q.ToList();
}
```

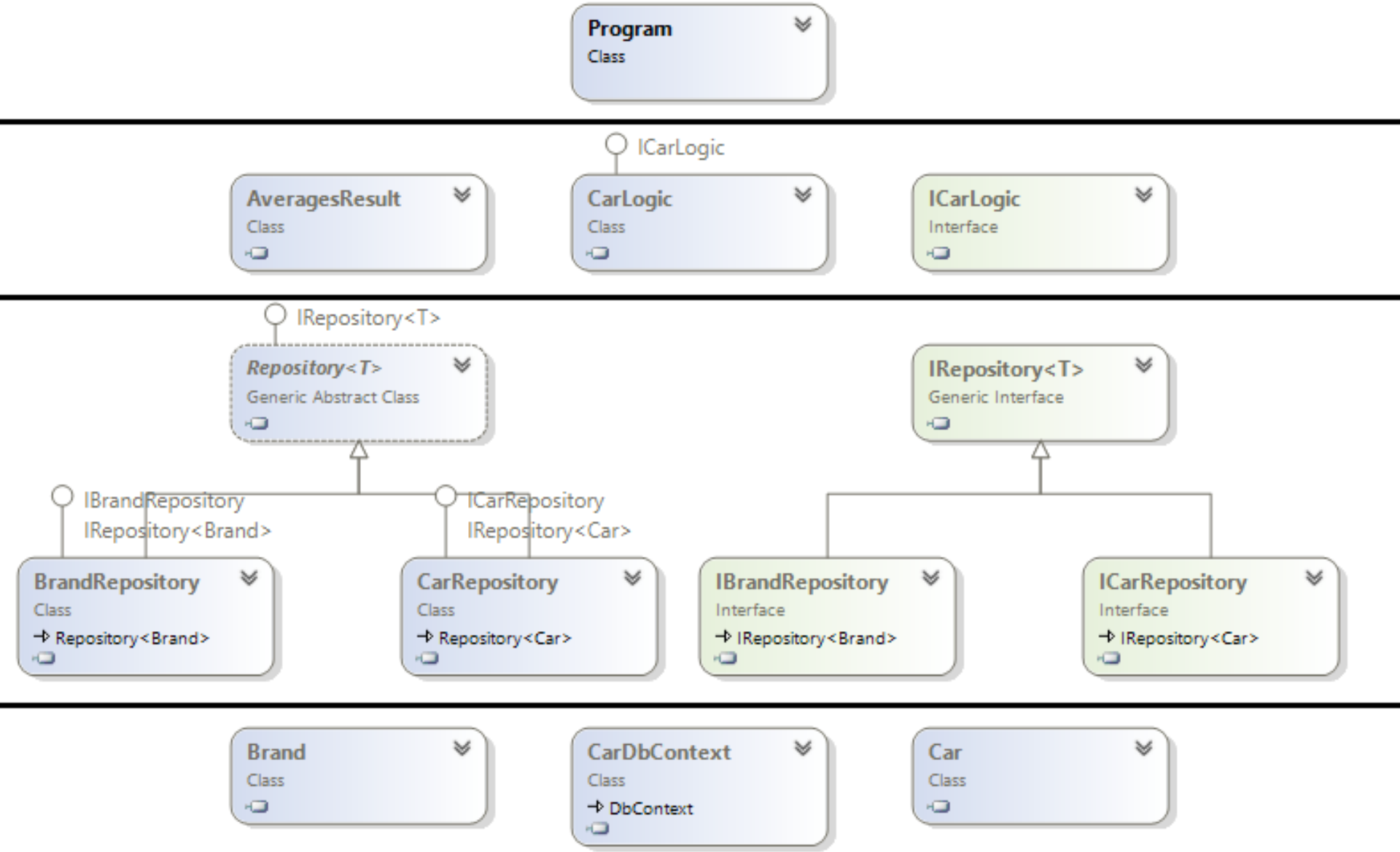
Rétegek: TEST

```
[Test] // Noncrud test
public void TestGetAveragesJoin()
{
    var logic = CreateLogicWithMocks();
    var actualAverages = logic.GetBrandAveragesJoin();

    Assert.That(actualAverages, Is.EquivalentTo(expectedAverages));
    carRepo.Verify(repo => repo.GetAll(), Times.Once);
    brandRepo.Verify(repo => repo.GetAll(), Times.Once);
}

[Test] // Crud Test: Add
public void TestBrandAdd()
{
    var logic = CreateLogicWithMocks();
    int idNumber = logic.AddBrand("Suzuki");
    Assert.That(idNumber, Is.EqualTo(42));
    brandRepo.Verify(repo => repo.Add(It.IsAny<string>()), Times.Once);
}
```

Rétegek: PROGRAM

- **A Console App csak Logic műveletet hív, a Logic a CRUD műveleteket továbbítja a Repo felé, és a Repo hívja a DbContext metódusokat.**
  - Minden réteg CSAK az alatta lévő réteggel kommunikálhat (esetleges felfele kommunikáció: eseményekkel - jelenleg nem szükséges)
- **Minden DbContext/Repository/Logic példányosítás itt történik, lehetőleg egy Factory osztályon keresztül**
  - DbContext leszármazottból EGYETLEN példányt használ mindegyik Repository (Singleton Design Pattern is, akár, hibalehetőségeket ld. Prog4)!
  - Logic osztályok ugyanazon Repo példányokat használják az egyes típusokból
- **SOLID elveknek megfelelő**
  - A Console App-on kívül más project NEM HASZNÁLHAT Console.Read/Write műveleteket
  - A Logic és a Console App NEM HASZNÁLHAT dbContext metódusokat, ez egyedül a Repository-nak engedélyezett
- **Menüvezérelt: ConsoleMenu-simple / EasyConsole**

Rétegzett alkalmazás



Problémák a rétegek között

- **Biztonsági probléma lehet, ha az ORM-hez csatolt Entity példányok a Data/Repo réteg fölött is elérhetőek**
  - Pl. a logikában nincs eljárás egy dolgozó fizetésének módosítására, de van eljárás egy részleg áthelyezésére
  - EMP singleEmp = myLogic.GetEmp(7788);
    singleEmp.SAL = 10000;
    myLogic.RelocateDept(30, ”Budapest”);
  - A RelocateDept() –ben lévő SaveChanges() a fizetés módosítását is elmenti!
  - Ez **most a féléves feladatban** nem gond, valódi alkalmazásnál köztes Business Model / DTO osztályok használata javasolt
- **A rétegek közötti példány-továbbítás megoldásai:**
  - Funkcionalitás-korlátozott entity példányok (pl. setter nélkül)
  - Láthatósággal: rétegen belül az internal is látszik, rétegek között csak a public
  - Manuális konverzió (reflexió … Teljesítmény?)
  - Automata konverzió (AutoMapper … Production-Ready?)
- **Hibajelzés (réteg-specifikus kivételkezelés)**
  - InnerException / AggregateException

Rétegek belső feldarabolása

- **Jelenleg a féléves feladatban a Repository-ban kötelező**
  - Logic ➔ legyen valahogy feldarabolva, és „TÖKÉLETES” …
- **Minimális elvárások**
  - Spagetthi code / Big Ball Of Mud elkerülése ➔ ezért bontottuk rétegekre
  - EF + IRepository<T>/IEmpRepository + ILogic/IEmpLogic + Console
- **Probléma: a Repository osztály így jó is lehet, de a Logic…**
  - God Object: Túl sok felelősségi kör egy osztálynak
  - Tipikus tünet: hosszú osztály
  - Tipikus tünet: osztály túl sok konstruktor paraméterrel / függőséggel
  - Megoldás: Refactor to individual classes
- **EmpLogic, DeptLogic, PremiumCalculationLogic …**
  - Ravioli code: elméletileg kicsi is könnyen érthető osztályok, de a rendszer egészének megértése nehéz (… Repository is an AntiPattern??? ☹)
  - Lasagna Code: elméletileg és ránézésre rétegzett kód, gyakorlatilag a belső káosz és szövevényes kereszthivatkozások miatt kezelhetetlen
  - **FŐ CÉL:** Arany középút ➔ REUSEABLE CODE!

Git

- **Git repository**
  - Verziókövetés: tudjuk, hogy melyik sort ki/mikor szerkesztette
  - Local repository, remote repository, **commit, push, pull**, ~~*branch, fork, merge*~~
  - Tetszőleges GIT kliens (parancssori / Sourcetree / TortoiseGit / VS)
- **Jelenleg egy branch és egy user**
  - Jelenleg egyszerű „mirror”, több branch/user a következő félévben
  - Bitbucket.org , Private repository
  - Kötelezően hozzáadandó admin joggal: **oe_nik_prog**
  - E-Mail cím szerinti hozzáadás: **nikprog@iar.nik.uni-obuda.hu**
- **Git conflicts**
  - Merge esetén, ha azonos file-t módosít több nem merge-elt commit
  - Azonos file-ban, de különböző területre vonatkozó commitok esetén automatikusan megoldódhat (automatic merge)
  - Azonos területre vonatkozó commit esetén manuális: keep A/local/ours, keep B/remote/theirs, run mergetool (kdiff3, Meld, Kompare) ➔ Prog4
- **GITSTATS – Elvárás a TRUE kimenet, mindenhol forduló kód !!!**

Stylecop.Analyzers + Microsoft.CodeAnalysis.FxCopAnalyzers

- **ELVÁRÁS, hogy MINDEN solution-beli projektben benne legyenek**
- **Nem csak a kód futási eredménye/teljesítménye, hanem a kód stílusa/formázása/olvashatósága is fontos (~ coding guidelines)**
  - https://docs.microsoft.com/en-us/visualstudio/code-quality/code-analysis-for-managed-code-overview
  - Tagolás, sortörések, nevek, láthatóságok
  - Blokk jelzése MINDIG kapcsos zárójelekkel
  - Kommentezés: minden publikus tag fölött (osztály, metódus, tulajdonság)
  - Metódus-definíciók esetén paraméterek/eredmény magyarázata
  - Osztályok/metódusok/sorok hosszára tessék figyelni!
- **Javasolt a „zero violations” code, de nem kötelező**
  - Ne legyen túl sok suppression vagy alapvető szabályt elrejtő suppression
  - Javasolt nem utolsó pillanatra hagyni, sokkoló tud lenni a hirtelen megjelenő 8571 warning…
- **ELVÁRÁS a nulla warning és nulla build error!**
  - A GitStats-ot nem fogja érdekelni, hogy miért nem nulla…

Doxygen

- **A Stylecop miatt minden osztály/metódus/tulajdonság felett dokumentációs XML kommentek lesznek**
  - Egy elkészült metódus vagy osztály fölött „///”-t gépelve létrejön
  - A kitöltött XML dokumentáció az Intellisense-ben is megjelenik
  - Project Properties / Build részben engedélyezzük az XML documentation file-t
  - A kommentek és a kód (minden név) nyelve: **ANGOL**
- **A Doxygen ebből generál „fogyasztható” dokumentumot**
  - Alapértelmezetten HTML formátum (csak ez az elvárt)
  - HTML Help builder használatával könnyedén generálható CHM dokumentáció
  - LaTeX használatával könnyedén generálható PDF dokumentáció
- **Parancssori változattal vagy a Doxywizard GUI segítségével**
  - A public láthatóságú tagok mellett kötelező az internal láthatóságú tagok feldolgozásának bekapcsolása is (EXTRACT_PACKAGE + EXTRACT_STATIC)
  - A HTML kimenetet a GIT repository-ba rakjuk bele, egy commitban menjen fel az egész a leadás előtt néhány órával/nappal (Documentation/Doxygen/index.html és még sok file ugyanott)