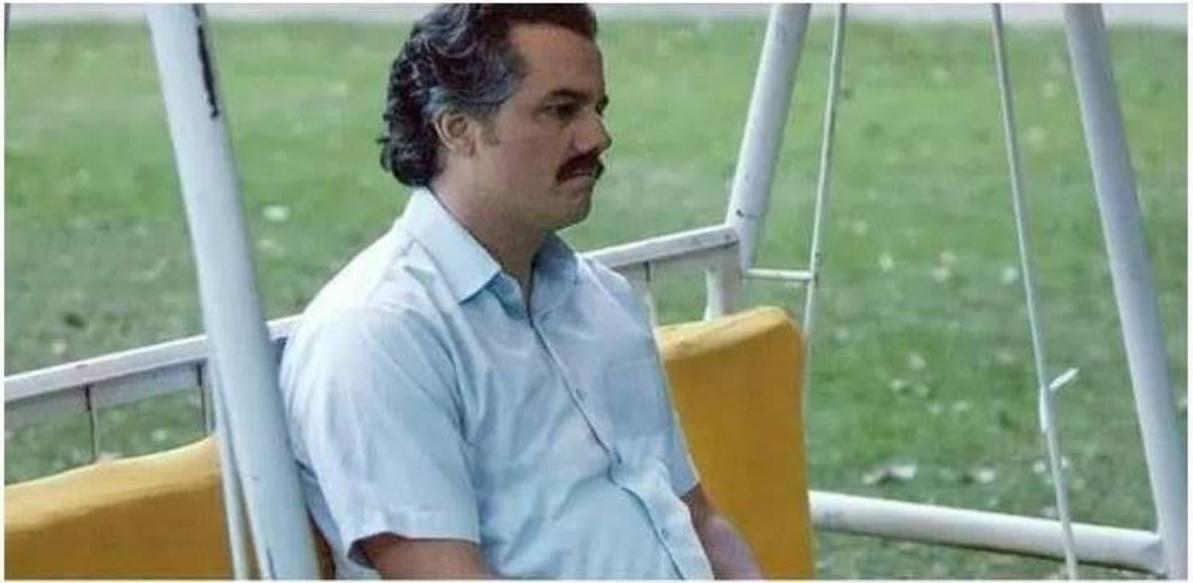


Uvod u softversko inženjerstvo

Značajni
nazivi

Nikola Luburić
nikola.luburic@uns.ac.rs

When you try to choose
a meaningful variable name.



Značaj održivog koda

Aspekti značajnih naziva

Značajni nazivi

Pomoć GPTa

Dve funkcije koje imaju isto vidljivo ponašanje

Značaj održivog koda

private List<CParameter> GetMethodParams1()

```
{  
    var memberParams = new List<CParameter>();  
    var paramLists = cSharpMember.Nodes().Type<ParameterSyntax>();  
    if (!paramLists.Any()) return memberParams;  
    var parameters = paramLists.First().Parameters;  
    foreach (var parameter in parameters)  
    {  
        var symbol = semanticModel.GetDeclaredSymbol(parameter);  
        memberParams.Add(new CParameter { Name = symbol.Name });  
    }  
    return memberParams;  
}
```

private List<CParameter> GetMethodParams2()

```
{  
    var paramLists = cSharpMember.Nodes().Type<ParameterSyntax>();  
    if (!paramLists.Any()) return new List<CParameter>();  
    return CreateCParams(paramLists.First().Parameters);  
}
```

Za prostu funkciju (npr. 5-10 linija koda) postoji bar 10 kombinacija programskih izraza koji daju isto **vidljivo ponašanje**

Ponašanje koda je skup efekata koji se manifestuju dok se kod izvršava

- Povratna vrednost izvršavanog koda,
- Izmena stanja aplikacije i njenih objekata,
- Promena podataka u eksternim sistemima,
- Upotreba radne memorije, procesora i drugih resursa računara...

Vidljivo ponašanje je podskup svog mogućeg ponašanja koje je interesantno posmatraču

- **Funkcionalna podobnost**,
- Performanse – metrike izvršavanja koda, na razumnoj skali,
- Bezbednost – nuspojave i ranjivosti...

Pisanje održivog koda je izbor skupa programskih izraza koji rezultuju željenim vidljivim ponašanjem i lako se održavaju

ISO 25010 razlaže održivost na:

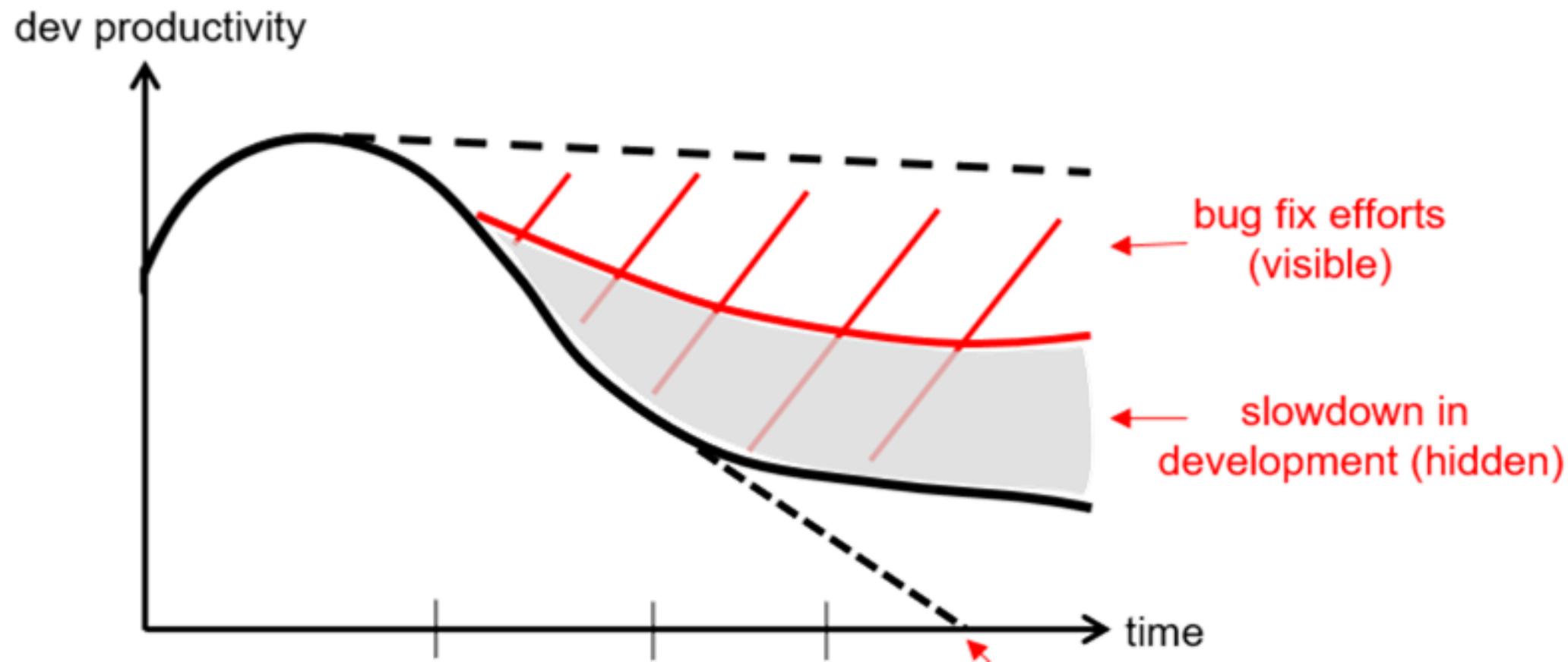
Maintainability

- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

- **Modularnost** - Stepen u kom je program sastavljen iz diskretnih modula tako da promena modula ima minimalan uticaj na ostale
- **Ponovna iskoristivost** - Mogućnost modula programa da se koriste na više mesta unutar sistema ili u više od jednog sistema
- **Analizabilnost** - Jednostavnost analize modula radi razumevanja rada, dijagnostikovanja uzroka otkaza ili planiranja izmene
- **Promenljivost** - Jednostavnost izmene modula bez uvođenja defekta ili umanjenja kvaliteta
- **Testabilnost** - Lakoća definisanja testnih kriterijuma za modul i provera njihove ispunjenosti

Pisanje održivog koda je izbor skupa programskih izraza koji rezultuju željenim vidljivim ponašanjem i lako se održavaju

Zapostavljanje održivosti stvara **tehnički dug**. Metafora duga ističe trošak koji se plaća da se otkloni dug i *kamatu* koja se redovno plaća dok tim radi



Pisanje održivog koda je izbor skupa programskih izraza koji rezultuju željenim vidljivim ponašanjem i lako se održavaju

Zapostavljanje održivosti stvara **tehnički dug**. Metafora duga ističe trošak koji se plaća da se otkloni dug i *kamatu* koja se redovno plaća dok tim radi

Tehnički dug otklanjamo kroz **refaktorisanja** ili **redizajniranja**

Refaktorisanje je usputna primena malog broja operacija refaktorisanja (npr. *rename*, *extract method*, *change method signature*) da se promeni struktura koda bez da se menja vidljivo ponašanje

Redizajniranje je krupna izmena koda koja zahteva mnoštvo operacija refaktorisanja kroz duži vremenski period

When the code is a mess
but it's working anyway



Burning Knight (Java branch)

This is not the latest branch of the game, see [this repo](#) for the C# branch.

You can read about why the game was rewritten in C# later on [here](#)

This is the Java branch of Burning Knight, the second version of the game (first one was written in Lua, but it only lasted for 3 days). I have to warn you, the code is not pretty. In fact, looking back at it, it's awful. But non the less, I've learned so much from this mess, and I tried my best to make the C# branch a lot better (and I think it worked out really well).

So. Have fun exploring this jungle. It might be a bit hard to compile, it requires some older gradle version, and I haven't figured out how to update the gradle build files to the latest syntax.



Available on [Steam](#) and [itch.io](#).

Before you look at the code: hear me out. Yes, some of it is not the best. Yes, some of it can be redone and improved. But I have to admit, I came so close to dropping this whole project so many times, I still have no idea, how I made it through May of 2020. Anyway. Here it is. The source code of C# branch of Burning Knight. For java branch see [this repo](#).

(Yes, this is only half of the work, the game was initially written in Java and then rewritten in C#, this code base is relatively nice compared to the Java one).

This project really quickly moved from the "nice and relaxing" to "constant bugfixing and stress" type of project. You can clearly see me going mad in some places (do not look at the pull requests I beg you). But its all in the past now.

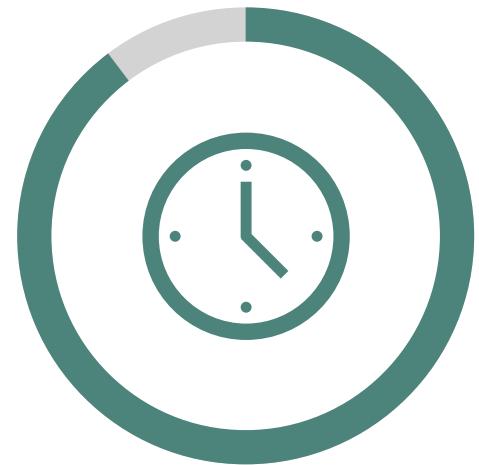
Koliko se kod čita, a koliko piše?

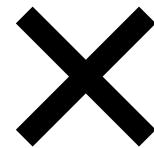
- ❖ Na zrelom projektu 10:1

- ❖ $devTime = R + W$

Čitljiv kod $R \rightarrow 0$

Aljkav kod $R \rightarrow \infty$

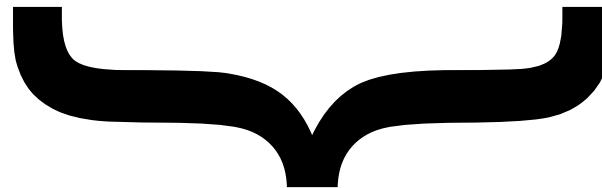




Održavanje košta

40 do 80%
ukupnog troška

Na zreлом projektu
odnos čitanja i
pisanja je 10:1



Većina troška razvoja je u čitanju koda
Održiv kod **značajno** smanjuje troškove



Michael Feathers

Working Effectively with Legacy Code

Clean code always looks like it was written by someone who cares. There is nothing obvious that you can do to make it better. All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you.

There are only two hard things in Computer Science: cache invalidation and naming things.

-- Phil Karlton

Nazivi prožimaju softver



Naziv treba da otkrije **svrhu** elementa

- Zašto postoji?
- Šta radi?
- Kako se koristi?

- ✓ 7. Konstruiši značajne nazive za identifikatore u kodu

- ✓ 1. Prati timske konvencije

- ✓ 2. Primeni prikladne tipove reči

- ✓ 3. Ukloni beznačajne reči

- ✓ 4. Koristi terminologiju domena problema

- ✓ 5. Analiziraj širi kontekst prilikom formiranja naziva

- ✓ 6. Formiraj naziv na dobrom nivou apstrakcije

Kako ih
velik tim
dogovara?

- ✓ 7. Konstruiši značajne nazive za identifikatore u kodu
 - ✓ 1. Prati timske konvencije
 - ✓ 2. Primeni prikladne tipove reči
 - ✓ 3. Ukloni beznačajne reči
 - ✓ 4. Koristi terminologiju domena problema
 - ✓ 5. Analiziraj širi kontekst prilikom formiranja naziva
 - ✓ 6. Formiraj naziv na dobrom nivou apstrakcije

- ✓ 7. Konstruiši značajne nazine za identifikatore u kodu
 - ✓ 1. Prati timske konvencije
 - ✓ 2. Primeni prikladne tipove reči
 - ✓ 3. Ukloni beznačajne reči
- ✓ 4. Koristi terminologiju domena problema
- ✓ 5. Analiziraj širi kontekst prilikom formiranja naziva
- ✓ 6. Formiraj naziv na dobrom nivou apstrakcije

Šta radi ovaj kod?

```
public List<int[]> GetThem() {  
    List<int[]> list = new List<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list.Add(x);  
    return list;  
}
```

šta se ovde
nalazi?

koji je značaj
ovoga (0 i 4)?

$R \rightarrow \infty$

šta da radim
sa ovim?

Šta radi ovaj kod?

```
public List<int[]> GetFlaggedCells() {  
    List<int[]> flaggedCells = new List<int[]>();  
    for (int[] cell : gameBoard)  
        if (cell[STATUS_VALUE] == FLAGGED)  
            flaggedCells.Add(cell);  
    return flaggedCells;  
}
```



Šta radi ovaj kod?

```
public List<Cell> GetFlaggedCells() {  
    List<Cell> flaggedCells = new List<Cell>();  
    for (Cell cell : gameBoard)  
        if (cell.IsFlagged())  
            flaggedCells.Add(cell);  
    return flaggedCells;  
}
```



✓ 7. Konstruiši značajne nazive za identifikatore u kodu

✓ 1. Prati timske konvencije

✓ 2. Primeni prikladne tipove reči

✓ 3. Ukloni beznačajne reči

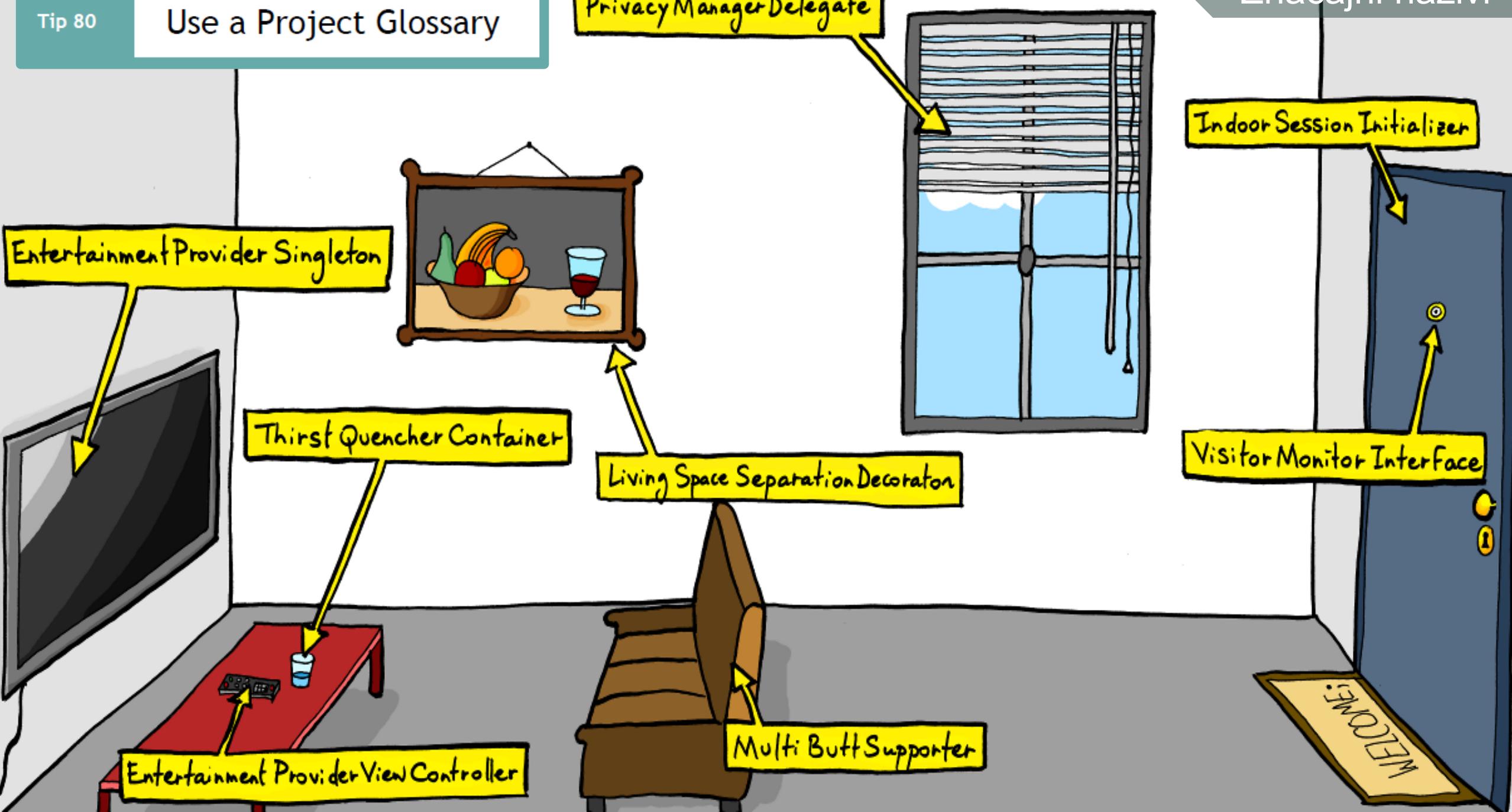
✓ 4. Koristi terminologiju domena problema

✓ 5. Analiziraj širi kontekst prilikom formiranja naziva

✓ 6. Formiraj naziv na dobrom nivou apstrakcije

Tip 80

Use a Project Glossary



✓ 7. Konstruiši značajne nazive za identifikatore u kodu

✓ 1. Prati timske konvencije

✓ 2. Primeni prikladne tipove reči

✓ 3. Ukloni beznačajne reči

✓ 4. Koristi terminologiju domena problema

✓ 5. Analiziraj širi kontekst prilikom formiranja naziva

✓ 6. Formiraj naziv na dobrom nivou apstrakcije

Šta su
primeri
konteksta?

// LearningTaskService.cs

```
LearningTaskFileRepository learningTaskFileRepository  
= new LearningTaskFileRepository();  
List<LearningTask> learningTasks  
= learningTaskFileRepository.GetAllLearningTasks();  
  
List<LearningTask> easyLearningTasks = new List<LearningTask>();  
foreach(LearningTask learningtask in learningTasks)  
{  
    if(learningTask.LearningTaskDifficulty > 1) continue;  
    easyLearningTasks.Add(learningTask);  
}  
  
return easyLearningTasks;
```

// LearningTaskService.cs

```
var repository = new TaskFileRepository();
```

```
var tasks = repository.GetAll();
```

```
List<LearningTask> easyLearningTasks = new List<LearningTask>();  
foreach(LearningTask learningtask in learningTasks)  
{  
    if(learningTask.LearningTaskDifficulty > 1) continue;  
    easyLearningTasks.Add(learningTask);  
}  
  
return easyLearningTasks;
```

```
// LearningTaskService.cs
```

```
var repository = new TaskFileRepository();
```

```
var tasks = repository.GetAll();
```

```
var easyTasks = new List<Task>();
```

```
foreach(var task in tasks)
```

```
{
```

```
    if(task.Difficulty > 1) continue;
```

```
    easyTasks.Add(task);
```

```
}
```

```
return easyTasks;
```

// LearningTaskService.cs

```
var repository = new TaskFileRepository();  
  
var tasks = repository.GetAll();  
  
var easyTasks = tasks.Where(t => t.Difficulty == 1).ToList();  
  
return easyTasks;
```

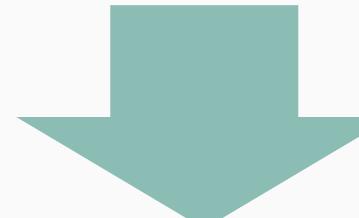
// LearningTaskService.cs

```
var repository = new TaskFileRepository();
```

```
var tasks = repository.GetAll();
```

```
return tasks.Where(t => t.Difficulty == 1).ToList();
```

```
// LearningTaskService.cs  
LearningTaskFileRepository learningTaskFileRepository  
    = new LearningTaskFileRepository();  
List<LearningTask> learningTasks  
    = learningTaskFileRepository.GetAllLearningTasks();  
List<LearningTask> easyLearningTasks = new List<LearningTask>();  
foreach (LearningTask learningtask in learningTasks)  
{  
    if (learningTask.LearningTaskDifficulty > 1) continue;  
    easyLearningTasks.Add(learningTask);  
}  
return easyLearningTasks;
```



```
var repository = new TaskFileRepository();  
var tasks = repository.GetAll();  
return tasks.Where(t => t.Difficulty == 1).ToList();
```

✓ 7. Konstruiši značajne nazive za identifikatore u kodu

✓ 1. Prati timske konvencije

✓ 2. Primeni prikladne tipove reči

✓ 3. Ukloni beznačajne reči

✓ 4. Koristi terminologiju domena problema

✓ 5. Analiziraj širi kontekst prilikom formiranja naziva

✓ 6. Formiraj naziv na dobrom nivou apstrakcije

Šta je nivo
apstrakcije?



7. Konstruiši značajne nazive za identifikatore u kodu

1. Opiši element koji imenuješ kroz rečenicu, koristeći termine iz domena problema ili rešenja.
2. Ukloni *noise* reči.
3. Ukloni reči koje se jednostavno izvlače iz šireg konteksta.
4. Analiziraj da li preostale reči opisuju koncept na dobrom nivou apstrakcije i da li su dobar tip reči.
5. Analiziraj da li preostale reči poštuju timske konvencije.
6. Postavi novi naziv.

Izazov

Cilj: Analiziraj kod i definiši unapređenja naziva identifikatora

1. Prođi kod i definiši šta svaki segment radi (*paper debugging*)
2. Pobroj identifikatore i njihove nazine
3. Spram razumevanja rada koda definiši bolje nazine

Diskutujte u grupicama od 2-3

Ne mogu da vas kontrolišem, morate sami

Posle 10 minuta nastavljamo, brzo smirivanje

Uticaj naziva na održivost?

Maintainability

- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

Analizabilnost (engl. *Analyzability*) ističe lakoću razumevanja strukture, ponašanja i namene koda pri planiranju izmena i dijagnostikovanja kvarova

**Značaj naziva je usko povezan
sa stepenom analizabilnosti koda**

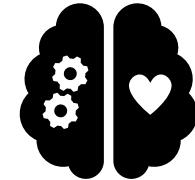
I am building an application and want to have meaningful names in my code. The application is similar to GetYourGuide, in that it has tours, tour guides and tourists. A tour has a name, description, location and other general information regarding the Tour. I have another concept that is related to the actual execution of a tour, which has a concrete tour guide, tourists that have signed up for the tour, and the exact time when the tour is starting.

To illustrate, a Tour would be a visit to the Vatican. This concept would include information where the Vatican is, what the general outline of the tour looks like, how long it lasts, etc. This other concept would include information that Bill will be the tour guide, that the tour will start at 9:00 and that 10 tourists have signed-up for the tour. This other concept would then track the status of the group (e.g., started, completed, in progress).

What is a good name for the concept?



vs



What would be a good name for the "Status" of the instance when the tour has not yet started? I can use "Active" when the tour is in progress and "Completed" when it is finished.