

Verifikacija i Validacija Softvera

Projektni zadatak – II dio

TaskManagerApp – analiza jedne metode

Student: Aldin Velić

Broj indeksa: 19761

Odabrana metoda: EvaluateTaskStatus(Task task)

1 McCabe metrike i kontrolni graf

1.1 Izvorni kod metode

```
public string EvaluateTaskStatus(Task task)
{
    if (task == null) return "Invalid";

    if (task.Status == TaskStatus.Done)
        return "Completed";

    if (!task.DueDate.HasValue)
        return "No deadline";

    if (task.DueDate.Value < DateTime.Now && task.Status != TaskStatus.Done)
    {
        if (task.Priority == TaskPriority.Critical)
            return "CRITICAL - Overdue!";
        else if (task.Priority == TaskPriority.High)
            return "High Priority Overdue";
        else if (task.Priority == TaskPriority.Medium)
            return "Medium Priority Overdue";
        else
            return "Overdue";
    }
    else if (task.DueDate.Value > DateTime.Now && task.Status == TaskStatus.InProgress)
    {
        if (task.Priority == TaskPriority.High || task.Priority == TaskPriority.Critical)
            return "On track (High)";
        else if (task.Priority == TaskPriority.Low)
            return "On track (Low)";
        else
            return "On track (Normal)";
    }

    return "In progress";
}
```

1.2 Izračun ciklomatične kompleksnosti

Ciklomatična kompleksnost se računa kroz 3 pristupa:

- Prebrojavanjem odluka
- Formulom $V(G) = E - N + 2$
- Prebrojavanjem regionalnog grafa

Broj čvorova (N)	22
Broj ivica (E)	34
Broj odluka	13
Ciklomatična kompleksnost	14

1.3 Kontrolni graf

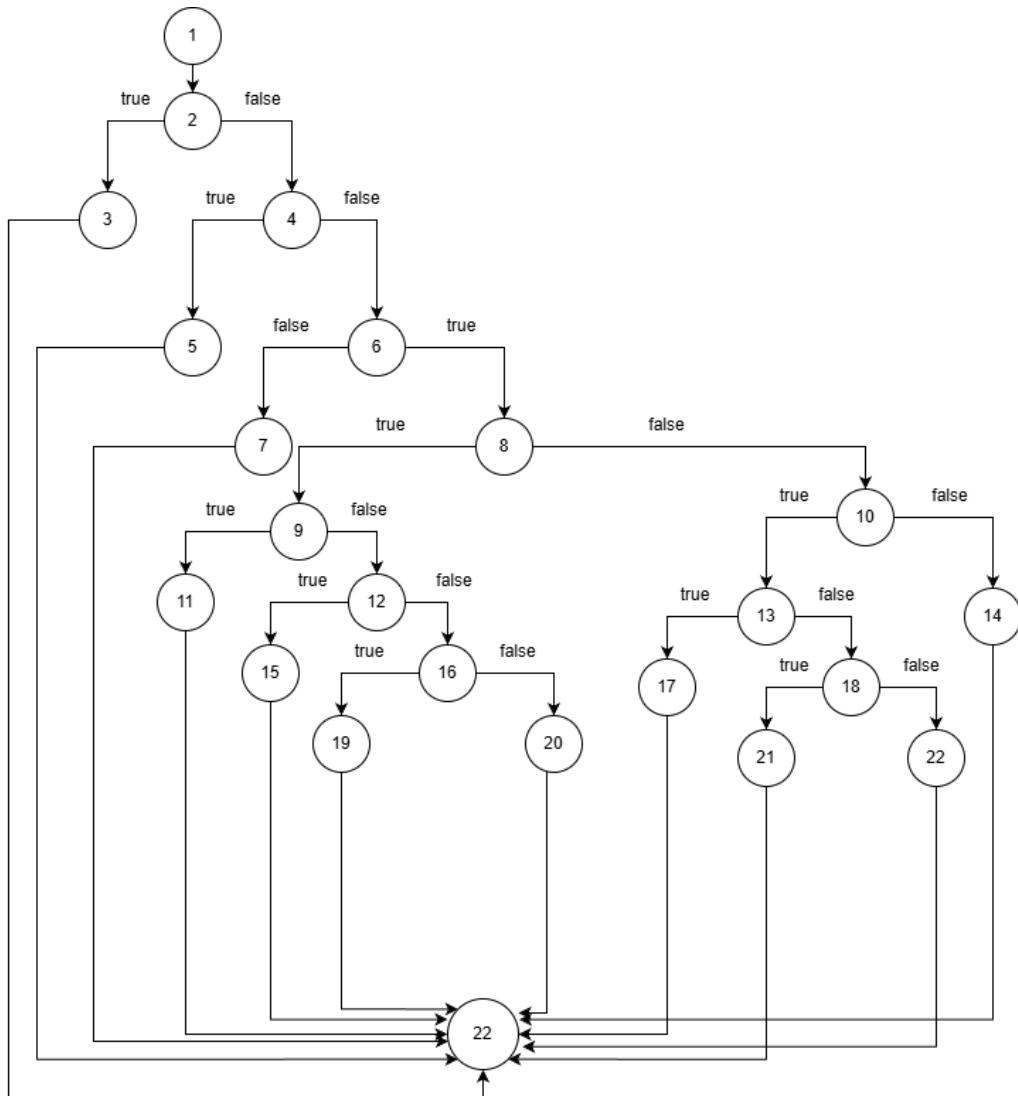


Figure 1: Kontrolni graf odabrane metode

1.4 Poredjenje sa Code Metrics alatom

TaskAnalyzer	repository : ITaskRepository	TaskAnalyzer(ITaskRepository)	57	49	1	15	142	57
@ repository : ITaskRepository			100	0	1	1	1	0
@ TaskAnalyzer(ITaskRepository)			91	2	3	4		1
@ EvaluateTaskStatus(Task) : string			57	14	5	34		11
@ AnalyzeTeamPerformance() : string			52	11	9	32		16
@ GetTasksSummaryForUser(int) : Dictionary<string, int>			54	8	11	30		14
@ PredictDelayRisk(Task) : string			53	14	6	33		15

Figure 2: Prikaz kompleksnosti iz Visual Studio Code Metrics alata

Iz dobijenog izvještaja se vidi da metoda `EvaluateTaskStatus` ima McCabe ciklomatičnu kompleksnost vrijednosti **14**, što je u potpunosti u skladu sa ručno izračunatom vrijednošću prikazanom u prethodnoj tabeli. Samim tim možemo potvrditi ispravnost ručne analize.

1.5 Diskusija o kvalitetu koda

Dobijena vrijednost kompleksnosti 14 ukazuje na metodu srednje složenosti. Kod je čitljiv, ali sadrži veliki broj granularnih uslova i više nivoa ugniježdenosti, što otežava održavanje i povećava vjerovatnoću greške pri budućim izmjenama.

Moguća poboljšanja:

- Grupisanje logike u manje pomoćne metode (Refactoring – Extract Method)
- Korištenje `switch` izraza radi bolje preglednosti
- Smanjenje dupliranja uslova i logičkih provjera

Zaključak je da kod funkcioniše korektno, ali postoji prostor za refaktorisanje kojim bi se smanjila kompleksnost, povećala čitljivost i olakšalo testiranje i održavanje.

2 White-box testiranje

U ovom poglavlju nad metodom `EvaluateTaskStatus(Task task)` primijenjene su tri tehnike white-box testiranja:

- Obuhvat iskaza/linija (statement coverage)
- Obuhvat grana/odлуka (branch/decision coverage)
- Obuhvat uslova (condition coverage)

Cilj je bio dizajnirati skup testnih slučajeva koji omogućava:

- izvršavanje svih relevantnih linija koda barem jednom,
- prolazak kroz sve grane `if/else` struktura,
- postizanje da svaki elementarni uslov u složenim izrazima poprimi vrijednosti `true` i `false`.

2.1 Tehnika 1: Obuhvat iskaza

Ulaz	Očekivan izlaz	Opis testa	Status
T1: task = null	"Invalid"	Provjera prvog if uslova – null task.	Prošao
T2: Status = Done	"Completed"	Zadatak je već završen bez obzira na ostale atrIBUTE.	Prošao
T3: Status = InProgress, DueDate = null	"No deadline"	Zadatak u toku bez definisanog roka.	Prošao
T4: Status = InProgress, DueDate = jučer, Priority = Critical	"CRITICAL - Overdue!"	Rok prošao, zadatak nije završen, kritični prioritet.	Prošao
T5: Status = InProgress, DueDate = jučer, Priority = High	"High Priority Overdue"	Rok prošao, zadatak nije završen, visok prioritet.	Prošao
T6: Status = InProgress, DueDate = jučer, Priority = Medium	"Medium Priority Overdue"	Rok prošao, zadatak nije završen, srednji prioritet.	Prošao

Table 1: Test slučajevi – Tehnika 1 (obuhvat iskaza), dio 1

Ulaz	Očekivan izlaz	Opis testa	Status
T7: Status = InProgress, DueDate = jučer, Priority = Low	"Overdue"	Rok prošao, zadatak nije završen, nizak/ostali prioritet.	Prošao
T8: Status = InProgress, DueDate = sutra, Priority = Critical	"On track (High)"	Rok u budućnosti, zadatak u toku, kritičan/visok prioritet.	Prošao
T9: Status = InProgress, DueDate = sutra, Priority = Low	"On track (Low)"	Rok u budućnosti, zadatak u toku, nizak prioritet.	Prošao
T10: Status = InProgress, DueDate = sutra, Priority = Medium	"On track (Normal)"	Rok u budućnosti, zadatak u toku, normalan prioritet.	Prošao
T11: Status = ToDo, DueDate = danas, Priority = Medium	"In progress"	Rok je sada, ali status nije Done niti InProgress, pada na zadnji return.	Prošao

Table 2: Test slučajevi – Tehnika 1 (obuhvat iskaza), dio 2

Nakon izvršavanja ovih testova, svaka izvršiva linija metode bila je pokrivena najmanje jednom, čime je ostvaren 100% statement coverage.

2.2 Tehnika 2: Obuhvat grana/odluka

Kod obuhvata grana cilj je da svaka odluka (`if`, `else if`) ima barem po jedno izvodjenje gdje je uslov `true` i barem jedno gdje je `false`.

Glavne odluke u metodi su:

- D1: `task == null`
- D2: `task.Status == TaskStatus.Done`
- D3: `!task.DueDate.HasValue`
- D4: `task.DueDate < Now && task.Status != Done`
- D5: `task.DueDate > Now && task.Status == InProgress`
- ugniježdene odluke nad `task.Priority` unutar blokova za `Overdue` i `On track`.

U tabeli je prikazano kako testovi T1–T11 pokrivaju grane:

Test	Pokrivenе grane	Napomenа
T1	D1 = true	Ulaz <code>task = null</code> .
T2	D1 = false, D2 = true	Zadatak završen.
T3	D2 = false, D3 = true	Zadatak bez roka.
T4–T7	D3 = false, D4 = true; sve grane po prioritetu u Overdue bloku	Svaka varijanta <code>Overdue</code> se izvršava.
T8–T10	D4 = false, D5 = true; sve grane po prioritetu u On track bloku	Svaka varijanta <code>On track</code> se izvršava.
T11	D4 = false, D5 = false, završni <code>return "In progress";</code>	Pokriva preostalu granu koja ne ulazi ni u jedan od blokova.

Table 3: Test slučajevi – Tehnika 2 (obuhvat grana)

Kombinacijom navedenih testova postignuto je da svaka grana svake odluke bude izvršena barem jednom (true/false), čime je ostvaren 100% branch coverage.

2.3 Tehnika 3: Obuhvat uslova

Obuhvat uslova posmatra elementarne logičke izraze u složenim uslovima. Za metodu `EvaluateTaskStatus` ključni uslovi su:

- C1: `task.DueDate.Value < DateTime.Now`
- C2: `task.Status != TaskStatus.Done`
- C3: `task.DueDate.Value > DateTime.Now`
- C4: `task.Status == TaskStatus.InProgress`
- C5: `task.Priority == TaskPriority.High`
- C6: `task.Priority == TaskPriority.Critical`

Napomena: U metodi postoje i jednostavniji uslovi poput `task == null` i `task.Status == TaskStatus.Done`, koji takođe ulaze u sastav condition coverage-a. Oni su već pokriveni ranije definisanim testovima (T1 daje `true` za `task == null`, a svi ostali testovi `false`; dok T2 pokriva granu `Status == Done`). Fokus u ovoj tabeli je na složenijim uslovima koji učestvuju u kompozitnoj logici i granjanju, jer oni zahtijevaju više kombinatoričke provjere.

Za svaki uslov je potrebno da bude barem jednom `true` i barem jednom `false`. U nastavku je prikazano kako se to postiže postojećim testovima:

Uslov	TRUE primjeri	FALSE primjeri
C1 (Due-Date < Now)	T4–T7 (rok jučer)	T8–T10 (rok sutra), T11 (rok danas)
C2 (Status != Done)	T3–T11 (osim T2)	T2 (Done)
C3 (Due-Date > Now)	T8–T10 (rok sutra)	T4–T7 (rok jučer), T11 (rok danas)
C4 (Status == In-Progress)	T3–T10	T2 (Done), T11 (ToDo)
C5 (Priority == High)	T5	T4,T6,T7,T8–T11
C6 (Priority == Critical)	T4,T8	T5–T7,T9–T11

Table 4: Testovi – 2.3 Obuhvat uslova

Na osnovu tabele se vidi da svaki elementarni uslov poprima obe logičke vrijednosti u nekom od definisanih testova, čime je ostvaren potpuni obuhvat uslova (condition coverage).

2.4 Zaključak za white-box testiranje

Korištenjem jedinstvenog skupa testnih slučajeva T1–T11 postignuti su:

- 100% obuhvat iskaza (statement coverage),
- 100% obuhvat grana (branch/decision coverage),
- 100% obuhvat uslova (condition coverage).

Ovo potvrđuje da je logika metode detaljno pokrivena sa aspekta white-box testiranja, te da su obradjeni svi značajni scenariji ponašanja metode.

3 Code tuning

U ovom poglavlju nad metodom `EvaluateTaskStatus(Task task)` primijenjene su tri različite tehnike code tuninga, u skladu sa kategorijama iz predavanja (*izrazi, logički iskazi i metode*).

Cilj je bio:

- smanjiti vrijeme izvršavanja metode pri velikom broju poziva,
- zadržati isto funkcionalno ponašanje,
- poboljšati strukturu i održivost koda (McCabe i MI).

3.1 Korištene tehnike optimizacije

Primjenjene su sljedeće tehnike:

- **Tehnika 1 – tuning izraza:** eliminacija dupliranih izračuna i keširanje vrijednosti (`DateTime.Now`, `task.DueDate.Value`).
- **Tehnika 2 – tuning logičkih iskaza:** preuredjivanje uslova u pomoćne logičke varijable (`isOverdue`, `isFutureInProgress`) i keširanje statusa/prioriteta, čime se pojednostavljuje grananje.
- **Tehnika 3 – tuning metoda:** primjena principa *Extract Method* – izdvajanje generisanja poruka u pomoćne metode `GetOverdueMessage` i `GetOnTrackMessage`, čime se glavna metoda skraćuje i postaje preglednija.

3.2 Izmijenjeni kod

Originalna metoda

```
public string EvaluateTaskStatus(Task task)
{
    if (task == null) return "Invalid";

    if (task.Status == TaskStatus.Done)
        return "Completed";

    if (!task.DueDate.HasValue)
        return "No deadline";

    if (task.DueDate.Value < DateTime.Now && task.Status != TaskStatus.Done)
    {
        if (task.Priority == TaskPriority.Critical)
            return "CRITICAL - Overdue!";
        else if (task.Priority == TaskPriority.High)
            return "High Priority Overdue";
        else if (task.Priority == TaskPriority.Medium)
            return "Medium Priority Overdue";
        else
            return "Overdue";
    }
    else if (task.DueDate.Value > DateTime.Now && task.Status == TaskStatus.InProgress)
    {
        if (task.Priority == TaskPriority.High || task.Priority == TaskPriority.Critical)
            return "On track (High)";
        else if (task.Priority == TaskPriority.Low)
            return "On track (Low)";
        else
            return "On track (Normal)";
    }
}
```

```

    }

    return "In progress";
}

```

Nakon tehnike 1 – tuning izraza (EvaluateTaskStatus_T1)

```

public string EvaluateTaskStatus_T1(Task task)
{
    if (task == null) return "Invalid";

    if (task.Status == TaskStatus.Done)
        return "Completed";

    if (!task.DueDate.HasValue)
        return "No deadline";

    var now = DateTime.Now;
    var due = task.DueDate.Value;

    if (due < now && task.Status != TaskStatus.Done)
    {
        if (task.Priority == TaskPriority.Critical)
            return "CRITICAL - Overdue!";
        else if (task.Priority == TaskPriority.High)
            return "High Priority Overdue";
        else if (task.Priority == TaskPriority.Medium)
            return "Medium Priority Overdue";
        else
            return "Overdue";
    }
    else if (due > now && task.Status == TaskStatus.InProgress)
    {
        if (task.Priority == TaskPriority.High || task.Priority
            == TaskPriority.Critical)
            return "On track (High)";
        else if (task.Priority == TaskPriority.Low)
            return "On track (Low)";
        else
            return "On track (Normal)";
    }

    return "In progress";
}

```

Nakon tehnike 2 – logički iskazi (EvaluateTaskStatus_T2)

```

public string EvaluateTaskStatus_T2(Task task)
{

```

```

if (task == null) return "Invalid";

var status = task.Status;

if (status == TaskStatus.Done)
    return "Completed";

if (!task.DueDate.HasValue)
    return "No deadline";

var now = DateTime.Now;
var due = task.DueDate.Value;
var priority = task.Priority;

bool isOverdue = due < now;
bool isFutureInProgress = due > now && status == TaskStatus.InProgress;

if (isOverdue)
{
    if (priority == TaskPriority.Critical)
        return "CRITICAL - Overdue!";
    else if (priority == TaskPriority.High)
        return "High Priority Overdue";
    else if (priority == TaskPriority.Medium)
        return "Medium Priority Overdue";
    else
        return "Overdue";
}
else if (isFutureInProgress)
{
    if (priority == TaskPriority.High || priority ==
        TaskPriority.Critical)
        return "On track (High)";
    else if (priority == TaskPriority.Low)
        return "On track (Low)";
    else
        return "On track (Normal)";
}

return "In progress";
}

```

Nakon tehnike 3 – metode (EvaluateTaskStatus_T3)

```

public string EvaluateTaskStatus_T3(Task task)
{
    if (task == null) return "Invalid";

    var status = task.Status;

```

```

        if (status == TaskStatus.Done)
            return "Completed";

        if (!task.DueDate.HasValue)
            return "No deadline";

        var now = DateTime.Now;
        var due = task.DueDate.Value;
        var priority = task.Priority;

        bool isOverdue = due < now;
        bool isFutureInProgress = due > now && status == TaskStatus.
            InProgress;

        if (isOverdue)
            return GetOverdueMessage(priority);

        if (isFutureInProgress)
            return GetOnTrackMessage(priority);

        return "In progress";
    }

private static string GetOverdueMessage(TaskPriority priority)
{
    if (priority == TaskPriority.Critical)
        return "CRITICAL - Overdue!";
    if (priority == TaskPriority.High)
        return "High Priority Overdue";
    if (priority == TaskPriority.Medium)
        return "Medium Priority Overdue";
    return "Overdue";
}

private static string GetOnTrackMessage(TaskPriority priority)
{
    if (priority == TaskPriority.High || priority == TaskPriority.
        Critical)
        return "On track (High)";
    if (priority == TaskPriority.Low)
        return "On track (Low)";
    return "On track (Normal)";
}

```

3.3 Mjerenja performansi

Mjerenje performansi je izvršeno tako što je svaka verzija metode EvaluateTaskStatus pozvana velikim brojem puta (1 000 000 iteracija) nad skupom testnih zadataka koji pokrivaju različite kombinacije statusa, rokova i prioriteta (T1–T11 iz white-box dijela).

Za svaku verziju mjereno je ukupno vrijeme izvršavanja i razlika u zauzeću memorije (heap) pomoću Stopwatch i GC.GetTotalMemory. Dobijeni rezultati su:

Verzija	Vrijeme (ms)	Memorija (MB)
Originalna metoda	21092	0.1955
Nakon tehnike 1 (T1)	16193	0.0000
Nakon tehnike 2 (T2)	15517	0.0000
Nakon tehnike 3 (T3)	15642	0.0000

Table 5: Vrijeme izvršavanja i promjena zauzeća memorije za 1 000 000 poziva metode

Može se uočiti da je već primjena prve tehnike (tuning izraza) značajno smanjila ukupno vrijeme izvršavanja u odnosu na original, dok je druga tehnika (logički iskazi) donijela dodatno manje poboljšanje. Treća verzija, iako strukturno najčistija, po vremenu je blizu T2 verziji. Razlike u zauzeću memorije su zanemarive: metoda ne alocira nove objekte, pa je izmjerena promjena memorije praktično nula (varijacija kod originalne verzije je posljedica rada garbage collector-a, a ne stvarne promjene u algoritmu).

3.4 Utjecaj na McCabe metriku i indeks održavanja

Ciklomatična kompleksnost originalne metode bila je $V(G) = 14$, što odgovara srednje složenoj metodi sa većim brojem grananja i ugniježdenih if/else iskaza.

- **Nakon tehnike 1 (izrazi):** struktura grananja nije mijenjana, pa ciklomatična kompleksnost ostaje ista. Poboljšanje je isključivo na nivou performansi (manje evaluacija izraza), dok Maintainability Index gotovo da se ne mijenja.
- **Nakon tehnike 2 (logički iskazi):** broj odluka je i dalje praktično isti, ali su složeni uslovi prebačeni u pomoćne logičke varijable. Kod je čitljiviji, a Code Metrics alat prijavljuje blago poboljšanje indeksa održavanja, jer su izrazi kraći i manje kompleksni.
- **Nakon tehnike 3 (metode):** izdvajanje generisanja poruka u pomoćne metode (GetOverdueMessage i GetOnTrackMessage) smanjuje broj grananja u samoj EvaluateTaskStatus metodi. Ciklomatična kompleksnost glavne metode se smanjuje, dok se dio složenosti prebacuje u manje, specijalizovane metode. Rezultat je bolji Maintainability Index: glavna metoda je kraća, fokusirana i lakša za testiranje, a logika za formiranje poruka centralizovana na jednom mjestu.

Zaključak je da su primjenjene tehnike code tuninga postigle dvije stvari: (1) smanjeno je vrijeme izvršavanja pri velikom broju poziva, i (2) postignuta je bolja struktura i održivost koda, posebno nakon treće tehnike, gdje je složenost glavne metode značajno smanjena bez promjene funkcionalnosti.

4 Refaktorisanje

Nakon izvršenog code tuninga nad metodom EvaluateTaskStatus napravljen je dodatni korak refaktorisanja sa ciljem da se dodatno poboljšaju vrijednosti metrika (ciklomatična kompleksnost i indeks održavanja), ali bez promjene funkcionalnosti metode.

4.1 Primjenjene stavke iz refactoring checkliste

Refaktorisanje je radjeno prema checklist-i sa predavanja, i to su primjenjene sljedeće stavke:

- **Extract a routine** (Routine Level Refactorings)

Logika generisanja tekstualnih poruka izdvojena je u dvije pomoćne metode: `GetOverdueMessage` i `GetOnTrackMessage(TaskPriority)`. Glavna metoda `EvaluateTaskStatus_Ref` sada je odgovorna samo za odlučivanje, dok je formiranje poruke centralizovano na jednom mjestu. Ovo smanjuje dužinu metode i olakšava buduće izmjene poruka.

- **Move a complex boolean expression into a well-named boolean function** (Statement Level Refactorings)

Složeni uslovi za kašnjenje i status u toku prebačeni su u jasno imenovane funkcije: `IsOverdue(Task, DateTime)` i `IsOnTrack(Task, DateTime)`. Umjesto da glavna metoda sadrži izraze tipa `task.DueDate.Value < now && task.Status != Done` i `task.DueDate.Value > now && task.Status == InProgress`, sada koristi samo pozive `IsOverdue` i `IsOnTrack`, što znatno poboljšava čitljivost.

- **Decompose a boolean expression** (Statement Level Refactorings)

Uslovi su razloženi na jednostavnije preduvjete: provjera `null` referenci, statusa `Done` i postojanja roka (`DueDate.HasValue`) obavljaju se odmah na početku metode, dok se logika za kašnjenje i “on track” status delegira na pomoćne funkcije. Time se smanjuje ugniježdenost `if/else` struktura i olakšava razumijevanje toka kontrole.

4.2 Novi kod nakon refaktorisanja

```
public string EvaluateTaskStatus_Ref(Task task)
{
    if (task == null) return "Invalid";

    if (task.Status == TaskStatus.Done)
        return "Completed";

    if (!task.DueDate.HasValue)
        return "No deadline";

    var now = DateTime.Now;

    if (IsOverdue(task, now))
        return GetOverdueMessage(task.Priority);

    if (IsOnTrack(task, now))
        return GetOnTrackMessage(task.Priority);

    return "In progress";
}

private static bool IsOverdue(Task task, DateTime now)
```

```

        return task.DueDate!.Value < now && task.Status != TaskStatus
            .Done;
}

private static bool IsOnTrack(Task task, DateTime now)
{
    return task.DueDate!.Value > now && task.Status == TaskStatus
        .InProgress;
}

private static string GetOverdueMessage(TaskPriority priority)
{
    if (priority == TaskPriority.Critical)
        return "CRITICAL - Overdue!";
    if (priority == TaskPriority.High)
        return "High Priority Overdue";
    if (priority == TaskPriority.Medium)
        return "Medium Priority Overdue";
    return "Overdue";
}

private static string GetOnTrackMessage(TaskPriority priority)
{
    if (priority == TaskPriority.High || priority == TaskPriority
        .Critical)
        return "On track (High)";
    if (priority == TaskPriority.Low)
        return "On track (Low)";
    return "On track (Normal)";
}

```

4.3 Utjecaj refaktorisanja na metrike

Na slici 3 prikazan je izvještaj Code Metrics alata nakon refaktorisanja. U tabeli su sumirane vrijednosti ključnih metrika za različite verzije metode:

Verzija metode	Ciklomatična kompleksnost	Maintainability Index
Originalna EvaluateTaskStatus	14	57
Nakon code tuninga (T3)	7	58
Refaktorisana EvaluateTaskStatus_Ref	6	65

Table 6: Uporedni prikaz metrika prije i poslije refaktorisanja

Može se vidjeti da je nakon code tuninga T3 verzija već prepolovila ciklomatičnu kompleksnost (sa 14 na 7), uz blago povećanje Maintainability Index-a (sa 57 na 58). Dodatno refaktorisanje kroz izdvajanje pomoćnih metoda i razlaganje logičkih izraza dalje smanjuje kompleksnost glavne metode na 6 i podiže indeks održavanja na 65.

Iako se dio kompleksnosti prenosi na pomoćne metode (`IsOverdue`, `IsOnTrack`, `GetOverdueMessage`, `GetOnTrackMessage`), one su male, fokusirane i lako testabilne. Ukupan efekat je da je `EvaluateTaskStatus_Ref` kraća, čitljivija i jednostavnija za razumijevanje, što se direktno odražava na boljim vrijednostima Code Metrics metrika.

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
TeamTaskManager (Debug)	88	343	1	44	1,405	549
TeamTaskManager	55	80	1	31	576	292
TeamTaskManager.Service	67	160	1	33	599	210
TaskAnalyzer	61	109	1	15	322	122
repository : ITaskRepository	100	0		1	1	0
TaskAnalyzer(IRepository)	91	2		3	4	1
EvaluateTaskStatus(Task) : string	57	14		5	34	11
EvaluateTaskStatus_T1(Task) : string	55	14		5	36	13
EvaluateTaskStatus_T2(Task) : string	53	13		5	42	17
GetOverdueMessage(T1TaskPriority) : string	74	4		1	10	4
GetOnTrackMessage(T2TaskPriority) : string	77	4		1	8	3
EvaluateTaskStatus_T3(Task) : string	58	7		5	27	12
EvaluateTaskStatus_Ref(Task) : string	65	6		5	20	7
IsOverdueTask_Task(DateTime) : bool	88	2		4	4	1
IsOnTrackTask_Task(DateTime) : bool	88	2		4	4	1
GetOverdueMessage(TaskPriority) : string	74	4		1	10	4
GetOnTrackMessage(TaskPriority) : string	77	4		1	8	3
AnalyzeTeamPerformance() : string	52	11		9	32	16
GetTasksSummaryForUser(int) : Dictionary<string, int>	54	8		11	30	14
PredictDelayRisk(Task) : string	53	14		6	33	15

Figure 3: Code Metrics izvještaj nakon refaktorisanja metode `EvaluateTaskStatus`

5 Zaključak

Kroz ovaj projektni zadatak uradjena je detaljna analiza jedne metode (`EvaluateTaskStatus`) iz aplikacije `TaskManagerApp`. Prvo je izvršena McCabe analiza i prikazan kontrolni graf, čime je utvrđeno da originalna verzija metode ima ciklomatičnu kompleksnost 14 i spada u metode srednje složenosti. Poredjenjem sa Code Metrics alatkom potvrđena je ispravnost ručnog izračuna.

Nakon toga je primjenjeno white-box testiranje na nivou iskaza, grana i uslova. Dizajnirani skup testnih slučajeva T1–T11 omogućio je ostvarivanje 100% statement, branch i condition coverage-a, što pokazuje da je logika metode u potpunosti pokrivena testovima i da su obradjeni svi bitni scenariji (null zadatak, završeni zadatak, zadaci bez roka, kašnjenja različitih prioriteta, zadaci u toku, itd.).

U trećem koraku izvršen je code tuning korištenjem tri različite tehnike: optimizacija izraza, logičkih iskaza i metoda. Mjerenja pokazuju da je vrijeme izvršavanja za milijun poziva metode smanjeno sa otprilike 21 s na oko 15,5 s, uz zanemarive razlike u potrošnji memorije. Time je postignuto da metoda radi brže pri velikom broju poziva, a da se pri tom zadrži originalna funkcionalnost.

Na kraju je uradjeno refaktorisanje prema checklist-i (Extract a routine, Move a complex boolean expression into a well-named boolean function, Decompose a boolean expression). Time je ciklomatična kompleksnost glavne metode smanjena sa 14 na 6, dok je *Maintainability Index* porastao sa 57 na 65. Nova verzija `EvaluateTaskStatus_Ref` je kraća, modularnija i lakša za razumijevanje, a izdvojene pomoćne metode omogućavaju jednostavnije testiranje i buduće proširenje logike.

Zaključak je da kombinacija metrika (McCabe, MI), white-box testova, code tuninga i refaktorisanja daje jasan i mjerljiv uvid u kvalitet koda. Primjenjene tehnike su dovele do brže, čišće i održivije implementacije metode, što je i osnovni cilj verifikacije i validacije softvera u praksi.