Załącznik nr 2

Studium przypadku: implementacja metod optymalizacji czasowo-kosztowej dla monitorowania poziomu efektywności w projekcie informatycznym[1]

Bartosz Kolonko

1. Opis przypadku

Przedmiotem tego **studium przypadku** (ang. *case study*) jest projekt przygotowania rozwiązania Microsoft Dynamics 365 dla klienta biznesowego oraz jego integracja z istniejącymi systemami ERP. Microsoft Dynamics 365 jest rozwiązaniem biznesowym dedykowanym dla firm o różnej wielkości i należy do klasy systemów ERP (ang. *Enterprise Resource Planning*) i CRM (ang. *Customer Relationship Management*). Rozwiązanie będzie osadzone w chmurze Microsoft Azure. Dodatkowo, w ramach integracji, przygotowana zostanie przestrzeń do przechowywania plików z Dynamics 365 w oparciu o Sharepoint. W fazie wstępnej projektu dokonano analizy systemowej i biznesowej na podstawie wymagań funkcjonalnych i pozafunkcjonalnych klienta oraz przygotowano dokumentację projektową określającą typy ryzyka, czasochłonność i zakres prac.

Początkiem prac deweloperskich w projekcie będzie stworzenie środowiska budowy i rozwoju, w tym środowiska na potrzeby testów **UAT** (ang. *User Acceptance Tests*) oraz środowiska produkcyjnego. W ramach implementacji powstaną elementy systemu Microsoft Dynamics 365 będące odwzorowaniem obiektów biznesowych i procesów, z którymi dotychczas pracował klient. Ostatnią częścią implementacji projektu będzie przygotowanie integracji Microsoft Dynamics 365 i Sharepoint, jako głównego systemu przechowującego pliki oraz istniejących systemów klienta ze środowiskiem Outlooka jako głównym komponentem do automatycznego generowania i przesyłania wiadomości mailowych.

[1] Na podstawie pracy magisterskiej Bartosza Kolonki *Optymalizacja czasowo-kosztowa projektów informatycznych*, promotor: Piotr Zaskórski, Warszawska Wyższa Szkoła Informatyki 2020.

Cały projekt będzie zakończony etapem przeprowadzenia testów w środowisku UAT oraz szkoleniem dla użytkowników i administratorów, a także wdrożeniem użytkowym połączonym ze wsparciem powdrożeniowym (nadzorem projektowym).

2. Założenia implementacyjne

Prezentowane studium przypadku jest ilustracją założeń i rozważań teoretycznych zawartych w rozdziale trzecim niniejszej publikacji. Głównym celem jest zatem pokazanie procesu optymalizacji czasowo-kosztowej przedstawionego wyżej projektu w oparciu o metodę CPM-COST, ze szczególnym odniesieniem do funkcjonalności tworzonej w tym celu aplikacji. W tym celu zakładamy, że spis wszystkich zadań w projekcie wraz z informacjami o ich ID, nazwami, informacjami o czasie trwania i granicznym czasie trwania zadań, danymi o koszcie normalnym i koszcie granicznym każdego zadania oraz o powiązaniach między zadaniami (poprzednikach zadania) znajdują w pliku Excel o nazwie „dane.xlsx" i są umieszczone w głównym folderze z aplikacją.

Tabela Z2.1. Fragment pliku „dane.xlsx" użytego jako dane wejściowe dla omawianej aplikacji

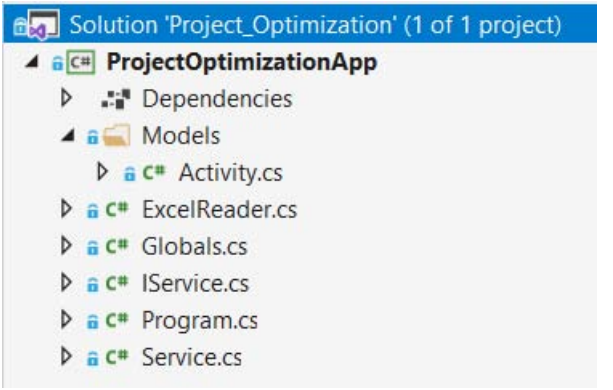
	A	B	C	D	E	F	G
1	Id	Description	Duratio	Termina	Cost	TerminalCos	Predecessor
2	1	Kick-off	0	0	0	0	
3	2	Opracowanie Planu Migracji	4	4	95	150	1
4	3	Analiza ryzyka projektu	4	4	95	150	2
5	4	Oszacowanie czasochłonności projektu	8	8	95	150	3
6	5	Opracowanie komponentu „Oferta"	16	16	95	150	4
7	6	Przygotowanie dokumentu - Plan projektu	16	16	95	150	5

Przedmiotowa aplikacja wspierająca zarządzanie czasem projektu została zaimplementowana w wybranym środowisku technologicznym.

3. Idea i implementacja aplikacji wspierającej zarządzanie projektami

Na podstawie dostępnych danych została zbudowana i zaimplementowana aplikacja w języku C# z użyciem framework.NET Core 3.1., wykorzystująca algorytm metody CPM-COST. Cała aplikacja bazuje na 6 klasach:

1) Program.cs – główna klasa całego programu (tutaj wywoływane są poszczególne metody odpowiedzialne za odczyt danych), opracowanie diagramu sieciowego wraz z wyznaczaniem ścieżki krytycznej i z wykorzystaniem procedury optymalizacji czasowo-kosztowej.
2) Activity.cs (wewnątrz folderu „Models") – model czynności w projekcie. Na podstawie tego modelu powstają obiekty, których wartości atrybutów czasowo-kosztowych są potem przeliczane w ramach metody CPM-COST.
3) Globals.cs – klasa ze zmiennymi globalnymi archiwizująca informacje o obecnym koszcie projektu oraz aktualny stan diagramu sieciowego na potrzeby metody CPM-COST. Cała klasa została zaimplementowana jako singleton.
4) ExcelReader.cs – klasa, której odpowiedzialnością jest odczyt pliku Excel i przygotowanie na podstawie otrzymanych danych listy obiektów typu Activity.
5) Service.cs – klasa zawierająca implementację wszystkich metod potrzebnych w ramach metody CPM-COST.
6) IService.cs – interfejs dla klasy Service.cs.



Rys. Z2.1 *Podgląd klas zawartych w ramach aplikacji*

Cała aplikacja realizuje zasadnicze trzy procesy:

1) wczytanie pliku z danymi;
2) zbudowanie diagramu sieciowego i wyznaczenie ścieżki krytycznej;
3) przeprowadzenie optymalizacji czasowo-kosztowej wg metody CPM-COST i wyliczenie zmian wektorów efektywności przy każdym kroku optymalizacji.

```
31 references
public class Activity
{
    9 references
    public int Id { get; set; }
    1 reference
    public string Description { get; set; }
    10 references
    public int Duration { get; set; }
    5 references
    public int TerminalDuration { get; set; }
    3 references
    public int Cost { get; set; }
    2 references
    public int TerminalCost { get; set; }
    6 references
    public int AverageCostGradient { get; set; }
    6 references
    public int EarliestStartTime { get; set; }
    7 references
    public int LatestStartTime { get; set; }
    8 references
    public int EarliestEndTime { get; set; }
    9 references
    public int LatestEndTime { get; set; }
    3 references
    public List<int> Predecessors { get; set; }
    2 references
    public List<int> Successors { get; set; } = new List<int>();
    2 references
    public bool IsCriticalPath { get; set; }
}
```

Rys Z2.2. *Klasa Activity*

Drugim procesem jest opracowanie diagramu sieciowego na podstawie wczytanych danych. Tutaj wykorzystujemy dwie metody: CalculateAhead i CalculateBackwards.

CalculateAhead (rys. Z2.3) służy do wyliczenia najwcześniejszych czasów rozpoczęcia i zakończenia poszczególnych zadań w projekcie. Całość rozpoczyna wyznaczenie najwcześniejszego czasu zakończenia dla pierwszego zadania w projekcie. Następnie dla wszystkich poprzedników każdego zadania w projekcie weryfikuje się, czy dany poprzednik kończy się później, niż zaczyna się dane zadanie. Jeśli tak, to najwcześniejszym czasem rozpoczęcia zadania staje się wartość odpowiadająca czasowi najwcześniejszego zakończenia realizacji poprzednika. Efektem jest wyliczenie czasu najwcześniejszego zakończenia zadania, przy czym: jeśli $\mathbf{EF}_A$ (ang. *Early Finish*) jest najwcześniejszą datą zakończenia poprzednika, to $\mathbf{ES}_B$ (ang. *Early Start*) dla następnika będzie równy $ES_B = EF_A + 1$ (dzień), gdzie $EF_A = ES_A + Duration_A$.

```
3 references
public List<Activity> CalculateAhead(List<Activity> list)
{
    list[0].EarliestEndTime = list[0].EarliestStartTime + list[0].Duration;

    for (int i = 1; i < list.Count; i++)
    {
        foreach (int actId in list[i].Predecessors)
        {
            Activity act = list.Find(x => x.Id == actId);

            if (list[i].EarliestStartTime < act.EarliestEndTime)
            {
                list[i].EarliestStartTime = act.EarliestEndTime;
            }
        }

        list[i].EarliestEndTime = list[i].EarliestStartTime + list[i].Duration;
    }

    return list;
}
```

Rys. Z2.3. *Metoda CalculateAhead*

CalculateBackwards (rys. Z2.4) zaczyna się od wyznaczenia najpóźniejszego czasu zakończenia ostatniego zadania w projekcie. W tym celu przyjmuje się wartość odpowiadającą najwcześniejszemu czasowi zakończenia ostatniego zadania. Kolejno obliczamy najpóźniejszy czas rozpoczęcia ostatniego zadania w projekcie – jako minimum czasów czynności wychodzących z danego węzła. Następnie, zaczynając od następników przedostatniego zadania i cofając się kolejno do poprzedniego zadania, weryfikujemy, czy mamy ustalony najpóźniejszy czas zakończenia zadania oraz najwcześniejszy czas rozpoczęcia tego zadania. Tak więc, jeśli EF_{ost} dla ostatniego zadania w analizowanym harmonogramie projektu wynosi $EF_{ost} = LF_{ost,}$ to **LS** (ang. *Late Start*) dla ostatniego zadania $LT_{ost} = LF_{ost}$ - $Duration_{ost}$, a więc **LF** (ang. *Late Finish*) dla zadania poprzedzającego będzie można obliczyć ze wzoru $LF_{poprzednik} = LS_{ost}$ - 1 (dzień).

```
3 references
public List<Activity> CalculateBackwards(List<Activity> list)
{
    var noOfElements = list.Count();

    list[noOfElements - 1].LatestEndTime = list[noOfElements - 1].EarliestEndTime;
    list[noOfElements - 1].LatestStartTime = list[noOfElements - 1].LatestEndTime - list[noOfElements - 1].Duration;

    for (int i = noOfElements - 2; i >= 0; i--)
    {
        foreach (int actId in list[i].Successors)
        {
            Activity act = list.Find(x => x.Id == actId);

            if (list[i].LatestEndTime == 0)
            {
                list[i].LatestEndTime = act.LatestStartTime;
            }
            else
            {
                if (list[i].LatestEndTime > act.LatestStartTime)
                {
                    list[i].LatestEndTime = act.LatestStartTime;
                }
            }
        }

        list[i].LatestStartTime = list[i].LatestEndTime - list[i].Duration;
    }

    return list;
}
```

Rys. Z2.4. *Metoda CalculateBackwards*

```
3 references
public void FindCriticalPath(List<Activity> list)
{
    Globals globals = Globals.GetState();

    foreach (Activity act in list)
    {
        if ((act.EarliestEndTime - act.LatestEndTime == 0) && (act.EarliestStartTime - act.LatestStartTime == 0))
        {
            act.IsCriticalPath = true;
            globals.criticalPath.Add(act.Id);
        }
    }

    int totalDuration = list[list.Count - 1].EarliestEndTime;

    if (globals.basicTotalDuration == 0)
    {
        globals.basicTotalDuration = totalDuration;
    }

    globals.currentTotalDuration = totalDuration;

    Console.WriteLine("Pierwotny czas trwania projektu: {0}", globals.basicTotalDuration);
    Console.WriteLine("Obecny czas trwania projektu: {0}", globals.currentTotalDuration);
}
```

Rys. Z2.5. *Metoda FindCriticalPath*

Fazą końcową (rys. Z2.5) tej procedury jest wyznaczenie ścieżki krytycznej. Tutaj szukamy zadań, dla których nie mamy zapasu czasu, czyli takich, gdzie rezerwa czasowa (zapas czasu) jest równa zero. Oznaczamy je jako krytyczne i zapisujemy do zmiennej, która później wykorzystywana jest do identyfikacji ścieżki krytycznej. Dodatkowo określamy pierwotny i aktualny czas potrzebny do wykonania projektu. Tak więc **całkowity zapas czasu dla zadania** (ang. *Total Float*) = |LS - ES| = |LF - EF|, przy czym:

LS – Late Start,
ES – Early Start,
LF – Late Finish oraz
EF – Early Finish.

Trzecim procesem (rys. Z2.6) jest przeprowadzenie optymalizacji czasowo-kosztowej. Zaczynamy od wyliczenia gradientu kosztu dla zadań, które możemy optymalizować (na potrzeby aplikacji zadania, których nie można zoptymalizować, mają takie same czasy trwania: normalny i graniczny). Zadania, których nie możemy zoptymalizować, mają ustalany gradient – jako nieokreślony (w aplikacji = -1).

```
2 references
public List<Activity> CalculateAverageCostGradient(List<Activity> list)
{
    foreach (var act in list)
    {
        if(act.Duration - act.TerminalDuration > 0)
        {
            act.AverageCostGradient = (act.TerminalCost - act.Cost) / (act.Duration - act.TerminalDuration);
        }
        else
        {
            act.AverageCostGradient = -1;
        }
    }

    return list;
}
```

Rys. Z2.6. *Metoda FindCriticalPath*

Kolejnym krokiem jest sama optymalizacja czasowo-kosztowa (rys. Z2.7). Wykonujemy ją dla zadań ulokowanych na ścieżce krytycznej, które możemy optymalizować (mają gradient inny niż -1) w kolejności od najmniejszego gradientu rosnąco. Dla optymalizowanego zadania zmieniamy jego czas trwania na ten równy granicznemu czasowi trwania i przeliczamy parametry dla wszystkich czynności w całym diagramie sieciowym, używając do tego metod CalculateAhead, CalculateBackwards i FindCriticalPath.

```
2 references
public void Optimize(List<Activity> activitiesData)
{
    Globals globals = Globals.GetState();

    foreach (var workspaceItem in globals.currentNetworkState.OrderBy(x => x.AverageCostGradient).ToList())
    {
        if (workspaceItem.IsCriticalPath && workspaceItem.AverageCostGradient != -1)
        {
            Console.WriteLine("\n");
            Console.WriteLine("Optymalizacja zadania o id = {0}", workspaceItem.Id);
            Console.WriteLine("Gradient kosztu: {0}", workspaceItem.AverageCostGradient);

            ClearCalculations(globals.currentNetworkState);

            workspaceItem.Duration = workspaceItem.TerminalDuration;

            globals.currentNetworkState = CalculateAhead(globals.currentNetworkState);
            globals.currentNetworkState = CalculateBackwards(globals.currentNetworkState);
            FindCriticalPath(globals.currentNetworkState);

            var originalAct = activitiesData.Where(x => x.Id == workspaceItem.Id).FirstOrDefault();

            globals.currentTotalCost += (originalAct.Duration - originalAct.TerminalDuration) * originalAct.AverageCostGradient;

            Console.WriteLine("Obecny koszt całkowity projektu: {0}", globals.currentTotalCost);

            CalculateEffectiveness();
        }
    }
}
```

Rys. Z2.7. *Metoda Optimize*

Na zakończenie tego procesu aktualizowany jest koszt projektu o wartość wynikłą z optymalizacji zadania i wyliczana jest zmiana wektora efektywności po dokonaniu optymalizacji zadania. W tym celu wylicza się efektywność (rys. Z2.8) jako iloraz zysku i kosztów (prognozowane zyski wprowadza się po uruchomieniu aplikacji), produktywność jako iloraz pierwotnego czasu trwania projektu i obecnego oraz zmianę kosztów w projekcie jako iloraz kosztów obecnych i pierwotnych wyrażonych w procentach.

```
1 reference
private void CalculateEffectiveness()
{
    Globals _globals = Globals.GetState();

    var effectiveness = _globals.expectedEarnings / _globals.currentTotalCost;

    double performance = _globals.basicTotalDuration / _globals.currentTotalDuration;

    var costChange = (_globals.currentTotalCost / _globals.basicTotalCost) - 1;

    Console.WriteLine("Efektywność projektu (stosunek spodziewanych zysków do kosztów projektu) : {0}", Math.Round(effectiveness, 2));
    Console.WriteLine("Wydajność projektu (stosunek pierwotnego czasu trwania projektu do obecnego czasu trwania projektu): {0}",
                                                                                    Math.Round(performance, 2));
    Console.WriteLine("Zmiana kosztu projektu względem pierwotnego kosztu: {0}", costChange.ToString("P", CultureInfo.InvariantCulture));
    Console.WriteLine("\n");
}
```

Rys. Z2.8. *Metoda Calculate Effectiveness*

Efektywność jest wyliczana po optymalizacji każdego zadania i jej zmiany są sygnalizowane po każdym kroku optymalizacji.

4. Specyfikacja i interpretacja wyników

W pierwotnych warunkach czas potrzebny na ukończenie projektu wynosi 613 godzin przy koszcie łącznym 55 985 zł. Spodziewany zysk po realizacji projektu został oszacowany na poziomie 95 000 zł. Po przeprowadzonych analizach i zbadaniu możliwości technologicznych ustalono plan optymalizacji czasowo-kosztowej projektu. Czasy realizacji poszczególnych zadań, graniczne czasy ich realizacji, koszt realizacji poszczególnych zadań wraz z granicznymi kosztami realizacji przedstawiono w tabeli Z2.2.

Tabela Z2.2. Zadania w projekcie po uwzględnieniu kosztów granicznych i czasów realizacji

ID	Nazwa zadania	Czas normalny trwania zadania (w godzinach)	Czas graniczny trwania zadania (w godzinach)	Koszt normalny wykonania zadania (zł/godzinę)	Koszt graniczny wykonania zadania (zł/godzinę)	Poprzednik
1	Kick-off	0	0	0	0	
2	Opracowanie Planu Migracji	4	4	95	150	1
3	Analiza ryzyka projektu	4	4	95	150	2
4	Oszacowanie czasochłonności projektu	8	8	95	150	3
5	Opracowanie komponentu „Oferta”	16	16	95	150	4
6	Przygotowanie dokumentu Plan Projektu	16	16	95	150	5
7	Przygotowanie dokumentu Zakres Prac	40	40	95	150	6
8	Przygotowanie dokumentów Kosztorys i Harmonogram	8	8	95	150	7
9	Project Management (10%)	7	7	95	150	8
10	Zebranie i analiza wymagań pozafunkcjonalnych	4	4	70	130	8
11	Analiza procesu sprzedażowego	8	8	100	215	8

ID	Nazwa zadania	Czas normalny trwania zadania (w godzinach)	Czas graniczny trwania zadania (w godzinach)	Koszt normalny wykonania zadania (zł/godzinę)	Koszt graniczny wykonania zadania (zł/godzinę)	Poprzednik
12	Analiza modelu danych i formularzy	16	16	100	215	11
13	Analiza modelu uprawnień – hierarchia organizacji	8	8	100	215	12
14	Analiza wymagań dot. integracji z systemem ERP klienta	8	8	100	215	13
15	Analiza wymagań dot. integracji z systemem do przechowywania dokumentów	4	4	100	215	14
16	Przygotowanie dokumentu Specyfikacja Wymagań	16	16	100	215	15
17	Poprawki do dokumentu Specyfikacja Wymagań	4	4	100	215	16
18	Akceptacja dokumentu specyfikacji wymagań przez klienta	2	2	100	215	17
19	Koniec etapu analizy przedwdrożeniowej	0	0	0	0	9;10;18
20	Project Management (20%)	70	70	95	150	19
21	Przygotowanie środowiska budowy i rozwoju	1	1	70	130	19
22	Konfiguracja środowiska budowy i rozwoju	2	2	70	130	21
23	Przygotowanie środowiska UAT	1	1	70	130	22
24	Konfiguracja środowiska UAT	2	2	70	130	23
25	Przygotowanie środowiska produkcyjnego	4	4	70	130	24

ID	Nazwa zadania	Czas normalny trwania zadania (w godzinach)	Czas graniczny trwania zadania (w godzinach)	Koszt normalny wykonania zadania (zł/godzinę)	Koszt graniczny wykonania zadania (zł/godzinę)	Poprzednik
26	Konfiguracja środowiska produkcyjnego	1	1	70	130	25
27	Konfiguracja integracji pomiędzy komponentami Dynamics 365 – środowisko budowy i rozwoju	4	4	70	130	26
28	Konfiguracja integracji pomiędzy komponentami Dynamics 365 – środowisko UAT	4	4	70	130	27
29	Konfiguracja integracji pomiędzy komponentami Dynamics 365 – środowisko produkcyjne	4	4	70	130	28
30	Odwzorowanie modelu danych w CRM	12	8	40	90	29
31	Przygotowanie encji „Klient”	2	1	40	90	30
32	Przygotowanie encji „Lead”	1	1	40	90	31
33	Przygotowanie encji „Szansasprzedaży”	2	1	40	90	32
34	Przygotowanie formularza UCI	4	2	40	90	33
35	Przygotowanie logiki ukrywania/ wyświetlania pól, walidacji wymagalności pól	8	6	40	90	34
36	Przygotowanie encji „Umowa”	4	3	40	90	35
37	Przygotowanie encji „Kontakt”	6	4	40	90	36
38	Przygotowanie encji „Prognoza przychodów”	2	2	40	90	37
39	Przygotowanie encji „Produkt”	2	2	40	90	38

ID	Nazwa zadania	Czas normalny trwania zadania (w godzinach)	Czas graniczny trwania zadania (w godzinach)	Koszt normalny wykonania zadania (zł/godzinę)	Koszt graniczny wykonania zadania (zł/godzinę)	Poprzednik
40	Przygotowanie encji „Produkt do Oferty”	2	2	40	90	39
41	Przygotowanie encji „Produkt do Umowy”	2	2	40	90	40
42	Przygotowanie encji „Dokument”	2	2	40	90	41
43	Przygotowanie logiki ukrywania / wyświetlania pól, walidacja wymagalności pól w zależności od typu dokumentu	4	3	40	90	42
44	Konfiguracja widoków	12	8	40	90	43
45	Konfiguracja formularza dokumentu w SharePoint	16	8	70	130	44
46	Stworzenie WF	16	10	70	130	45
47	Stworzenie BPF	16	10	70	130	46
48	Stworzenie walidacji wprowadzonych dokumentów	16	10	70	130	47
49	Plugin generowania prognozy przychodów	40	25	70	130	48
50	Konfiguracja produktów	4	3	70	130	49
51	Konfiguracja ról oraz uprawnień	4	4	70	130	50
52	Konfiguracja uprawnień procesowych	8	6	70	130	51
53	Stworzenie szablonów dokumentów / wiadomości e-mail	12	6	40	90	52
54	Integracja kartoteki klientów	40	25	100	215	53
55	Integracja Umowy	40	25	100	215	54

ID	Nazwa zadania	Czas normalny trwania zadania (w godzinach)	Czas graniczny trwania zadania (w godzinach)	Koszt normalny wykonania zadania (zł/godzinę)	Koszt graniczny wykonania zadania (zł/godzinę)	Poprzednik
56	Konfiguracja standardowej synchronizacji z systemem SharePoint	2	2	0	0	55
57	Konfiguracja jednej aplikacji UCI	4	4	70	130	56
58	Dashboard konwersji leadów	4	4	70	130	57
59	Dashboard Workplace (2 dashbordy)	8	8	70	130	58
60	Konfiguracja dodatku do Outlook	1	1	70	130	59
61	Przygotowanie dokumentu Plan Testów	16	16	70	130	60
62	Poprawki do dokumentu Plan Testów	6	6	70	130	61
63	Akceptacja dokumentu Plan Testów przez klienta	2	2	70	130	62
64	Przeniesienie rozwiązaniana UAT	4	4	70	130	63
65	Testy klienta	16	16	95	150	64
66	Poprawki po testach	8	8	70	130	65
67	Testy przedwdrożeniowe z klientem	8	8	95	150	66
68	Akceptacja raportu testów	2	2	95	150	67
69	Faza implementacji – zakończona	0	0	0	0	68
70	Szkolenie dla użytkowników	12	12	95	150	20; 69
71	Szkolenie dla administratorów	8	8	95	150	70
72	Faza stabilizacji – zakończona	0	0	0	0	70; 71
73	Przygotowanie planu deployu i recovery	2	2	70	130	72

ID	Nazwa zadania	Czas normalny trwania zadania (w godzinach)	Czas graniczny trwania zadania (w godzinach)	Koszt normalny wykonania zadania (zł/godzinę)	Koszt graniczny wykonania zadania (zł/godzinę)	Poprzednik
74	Wdrożenie na produkcję	8	8	70	130	73
75	5 dni asysty powdrożeniowej do realizacji w ciągu 3 tygodni od startu produkcyjnego rozwiązania	40	40	70	130	74
76	Uruchomienie produkcyjne – zakończone	0	0	0	0	75

Zaprezentowane powyżej dane zostały zamieszczone w pliku Excel i użyte jako dane wejściowe do aplikacji. W wyniku uruchomienia na tych danych aplikacji została wyznaczona ścieżka krytyczna dla tego projektu, która zawiera czynności/zadania o numerach 1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75 i 76. Następnie rozpoczął się iteracyjny proces optymalizacji czasowo-kosztowej kolejnych zadań, zaczynając od tych o najniższym gradiencie kosztów. Informacje o kolejnych krokach optymalizacji czasowo-kosztowej projektu zostały zawarte w tabeli Z2.3.

Tabela Z2.3. Wyniki optymalizacji czasowo-kosztowej projektu za pomocą aplikacji

ID zadania	Gradient kosztu	Czas trwania projektu po optymalizacji (w godzinach)	Koszt projektu po optymalizacji (w złotówkach)	Efektywność projektu	Wydajność projektu	Zmiana kosztu projektu względem pierwotnego kosztu
49	4	598	56 045	1,7	1,03	0,11%
45	7	590	56 101	1,69	1,04	0,21%
54	7	575	56 206	1,69	1,07	0,39%
55	7	560	56 311	1,69	1,09	0,58%
53	8	554	56 359	1,69	1,11	0,67%
46	10	548	56 419	1,68	1,12	0,78%
47	10	542	56 479	1,68	1,13	0,88%
48	10	536	56 539	1,68	1,14	0,99%
30	12	532	56 587	1,68	1,15	1,08%
44	12	528	56 635	1,68	1,16	1,16%
34	25	526	56 685	1,68	1,17	1,25%

ID zadania	Gradient kosztu	Czas trwania projektu po optymalizacji (w godzinach)	Koszt projektu po optymalizacji (w złotówkach)	Efektywność projektu	Wydajność projektu	Zmiana kosztu projektu względem pierwotnego kosztu
35	25	524	56735	1,67	1,17	1,34%
37	25	522	56785	1,67	1,17	1,43%
52	30	520	56845	1,67	1,18	1,54%
31	50	519	56895	1,67	1,18	1,63%
33	50	518	56945	1,67	1,18	1,71%
36	50	517	56995	1,67	1,19	1,80%
43	50	516	57045	1,67	1,19	1,89%
50	60	515	57105	1,66	1,19	2,00%

Biorąc pod uwagę powyższe dane widoczne jest, że dokonano redukcji czasu trwania projektu o 98 godzin, co stanowi przyśpieszenie o 19% względem pierwotnego czasu potrzebnego na realizację projektu. Całkowity koszt projektu po jego optymalizacji zwiększył się do 57 105 zł, co daje wzrost o 1120 zł względem pierwotnie zakładanego kosztu realizacji projektu, co jednocześnie jest widoczne w efektywności projektu, zmniejszając ją o 0,04.

Widzimy zatem, że wraz ze skróceniem czasu potrzebnego na realizację projektu zwiększał się jego koszt i zmniejszała jego efektywność w wymiarze operacyjnym. Postrzegając jednak projekt jako złożone przedsięwzięcie długookresowe, można spodziewać się, że zwiększone nakłady na sam proces projektowania (lepsze, wydajniejsze zasoby projektowe) mogą skutkować lepszym poziomem jakości i na etapie utrzymywania rozwiązań (nadzoru autorskiego) koszty nie będą wzrastać lub tempo ich wzrostu może być powolniejsze, co oznaczać może, że efektywność całkowita może mieć trend wzrostowy.