|  |  |
| --- | --- |
| ПРОЕКТ  „ИРИДА“  Разрбоване на детализирани училищни учебни програми за предметите от професионалната подготовка по специалност „Приложно програмиране“ |  |
|  |  |
| Банско 2025  инж. Георги Бориков |  |

Съдържание

[Увод 3](#_Toc202037705)

[Обзор на съществуващи методологии 4](#_Toc202037706)

[BOPPPS моделът – Структуриране на учебния процес 5](#_Toc202037707)

[Моделът „Постепенно освобождаване на отговорност“ (GRR) 5](#_Toc202037708)

[Принципът FLOW – Оптимално адаптиране в класната стая 6](#_Toc202037709)

[MSCW (или понякога MoSCoW) анализ 7](#_Toc202037710)

[Използване на автентични инструменти и технологии в приложното програмиране 8](#_Toc202037711)

[Методология (технология) 9](#_Toc202037712)

[Алгоритъм и ключови стъпки 9](#_Toc202037713)

[Откроени характеристики и предимства 11](#_Toc202037714)

[ Интеграция на съвременни принципи: 11](#_Toc202037715)

[ Приобщаващ и адаптивен подход: 11](#_Toc202037716)

[ Ефективна приоритизация: 11](#_Toc202037717)

[ Осигуряване на автентичност и мотивация: 11](#_Toc202037718)

[ Обобщение 11](#_Toc202037719)

[Реализация 12](#_Toc202037720)

[Стъпка 1: Анализ на рамковия учебен план 12](#_Toc202037721)

[Стъпка 2: Групиране по теория/практика и установяване на синхрон 13](#_Toc202037722)

[Стъпка 3: Определяне на учебните седмици 13](#_Toc202037723)

[Стъпка 4: Формулиране на цели и задачи по предмети 15](#_Toc202037724)

[Стъпка 5: MoSCoW анализ на съдържание 15](#_Toc202037725)

[Стъпка 6: Разбиване на програмата на занятия (уроци) 20](#_Toc202037726)

[Стъпка 7: Синхронизиране на теория и практика 22](#_Toc202037727)

[Стъпка 8: Разработване на уроците 22](#_Toc202037728)

[Заключение 28](#_Toc202037729)

[Източници 30](#_Toc202037730)

# Увод

Адаптирането на учебни програми е критичен процес за постигане на съответствие между стандартния рамков учебен план и специфичните нужди на учениците, както и на съвременните технологични изисквания на учебната институция. В областта на приложното програмиране, тази адаптация придобива за особен интерес с оглед на динамичната промяна на технологичните изисквания и постоянното развитие на софтуерни инструменти като Python и PyCharm. Този документ има за цел да представи систематизиран алгоритъм за анализ и адаптиране на рамковите учебни програми по „Приложно програмиране“, интегрирайки доказани методики, насочени към активното обучение, приобщаващото образование и използването на автентични технологични инструменти.

Учебните програми по приложно програмиране трябва да отразяват нуждите на различни групи ученици – от тези с по-големи технически умения до ученици със специфични образователни потребности. Този документ обхваща съвременните подходи, включително модела BOPPPS за планиране на уроци, модела GRR (Постепенно освобождаване на отговорност), принципа FLOW и метода MoSCoW за приоритизация, които позволяват динамична и адаптивна трансформация на традиционния учебен план. Освен това, се обръща специално внимание на използването на реални казуси и автентични инструменти, които обогатяват учебния процес и подготвят учениците за бъдещи предизвикателства в професионалната среда.

Адаптирането на учебни програми е ключов процес за осигуряване на съответствие между рамковите държавни изисквания, индивидуалните учебни потребности и актуалните технологични стандарти на съвременното училище. В областта на приложното програмиране тази необходимост е особено подчертана – поради бързо развиващите се технологии, регулярните промени в софтуерните инструменти (като Python и PyCharm) и необходимостта учениците да са гъвкави и подготвени за реален технологичен свят.

Настоящият документ си поставя няколко основни цели:

* Да предложи ясно структуриран и приложим алгоритъм за анализ, детайлизиране и адаптиране на рамковите учебни програми по „Приложно програмиране“ спрямо спецификата на конкретното училище и ученически профил;
* Да интегрира проверени образователни методики, които стимулират активно учене, приобщаваща среда и мотивирано участие – включително BOPPPS (или 5Е/5PPPS) модел за планиране на уроци, GRR (Постепенно освобождаване на отговорност), принципа FLOW и метода MoSCoW за приоритизация;
* Да подчертае важността на реалните технологични казуси и ежедневно използваните инструменти в обучението, така че учениците не просто да усвояват теоретични концепции, а да бъдат подготвени за практически професионални предизвикателства;
* Да представи примери за конкретно приложение на избраните подходи в училищна среда, вкл. варианти за диференцирана работа според нивото или нуждите на групата.

Документът съдържа обзор на ключови съвременни педагогически методологии с фокус върху тяхната приложимост в контекста на специалността „Приложно програмиране“, разгръща системна методология (алгоритъм) за детайлизиране на учебните програми и представя практически решения за реализация на адаптираните програми, базирани на нуждите на ПГЕЕ-Банско. Като краен резултат, се цели създаването на практична и устойчива основа за модернизирано професионално обучение, която може да служи както за пример, така и за база за по-нататъшно развитие и надграждане в системата на професионалното образование.

# Обзор на съществуващи методологии

**В контекста на специалността „Приложно програмиране“ възниква необходимост от адаптирани образователни подходи, които съчетават овладяването на фундаментални теоретични знания и развитието на практически умения. Предложеният подбор и интеграция на съвременни педагогически методологии е съобразен с динамиката на софтуерната индустрия и спецификата на професионалната подготовка, за да отговори максимално ефективно на нуждите на учениците и учителите.**

В тази секция се разглеждат установените методологии за адаптиране на рамковите учебни програми, които могат да бъдат интегрирани в учебната дисциплина „Приложно програмиране“ за постигане на максимална ефективност и приобщаващост.

## BOPPPS моделът – Структуриране на учебния процес

BOPPPS (Bridge-In, Outcome, Pre-Assessment, Participatory Learning, Post-Assessment, Summary) е структуриран метод за планиране на учебни моменти, който позволява активното включване на учениците още от началото на урока. В този модел:

* **Bridge-In:** Учителят установява контекста и възбужда интерес посредством реални примери от живота или професионалната практика, например нуждата от владеене на Python за решаване на конкретни задачи в софтуерната индустрия.
* **Outcome:** Ясно се дефинират целите на урока, като се поставят акценти върху придобиването на конкретни умения (напр. работа с PyCharm), свързани с приложното програмиране.
* **Pre-Assessment:** Провеждане на предварителна оценка, чрез кратки анкети или тестове, за да се установи нивото на знания и опит на учениците.
* **Participatory Learning:** Активно участие чрез практически упражнения, работа в екипи, разработка на проекти и използване на реални казуси.
* **Post-Assessment:** След приключване на урока се оценява постигнатото, както чрез обратна връзка, така и чрез тестове.
* **Summary:** Обобщаване на наученото в края на урока, което подпомага консолидирането на знанията.

Този модел позволява структурирана адаптация на всеки урок, като се отчита не само съдържанието, но и процесът на усвояване на материята, осигуряващ необходимата гъвкавост за интегриране на специални образователни нужди или технологии.

**Контекст на приложимост:** Подходящ за структуриране на отделните уроци по програмиране така, че във всяка стъпка учениците да са ангажирани, да разбират целите и да прилагат новите знания.

## Моделът „Постепенно освобождаване на отговорност“ (GRR)

Моделът GRR (Gradual Release of Responsibility) представлява насърчаване на развиването на уменията на учениците чрез постепенното намаляване на ролята на учителя и увеличаване на тяхната независимост. Този метод включва четири етапа:

1. Директно обучение: Учителят показва примери, демонстрира решения и обяснява процеси – например как да се конфигурира PyCharm или да се стартира Jupyter Notebook.
2. Ръководена практика: Учениците изпълняват задачи под наблюдението на учителя, който осигурява допълнителни обяснения при необходимост.
3. Сътрудничество при учене: Работа в малки групи или екипи, където учениците си помагат взаимно – отличен начин за интегриране на метода на екипно програмиране.
4. Самостоятелна работа: Учениците изпълняват задачите самостоятелно, прилагайки наученото на практика.

Този модел увеличава мотивацията и изграждането на увереност у учениците, като осигурява плавен преход от директна инструкция към самостоятелно овладяване на материала.

**Контекст на приложимост:** Изключително ефективен при въвеждане и усвояване на нови програмни концепции чрез постепенно преминаване от преподавателско ръководство към ученическа самостоятелност.

## Принципът FLOW – Оптимално адаптиране в класната стая

Принципът FLOW е насочен към създаване на среда, която „тече“ без усилия, като се интегрират следните критерии:

* **Fit (Уместност и приложимост):** Адаптацията трябва да бъде съобразена с конкретната класна стая и ресурсите, с които разполага тя.
* **Lend (Помощ и съдействие):** Методът трябва да отговаря на индивидуалните нужди на всеки ученик, като предлага допълнителни ресурси или обяснения, ако са необходими.
* **Optimize (Оптимизиране):** Стремеж към оптимално разбиране и усвояване на материала чрез практическо приложение и повторение.
* **Work (Сътрудничество):** Методът трябва да бъде съвместим с планираните учебни дейности и да насърчава групова работа и взаимопомощ.

Този подход гарантира, че адаптирането не само отговаря на учебните цели, но и се вписва хармонично в динамиката на класната стая.

**Контекст на приложимост:** Осигурява адаптиране на учебния процес и съдържание към конкретните характеристики на учебната среда и ученическите профили в специалността „Приложно програмиране“.

## MSCW (или понякога MoSCoW) анализ

MSCW (или MoSCoW) анализ, от английските думи: Must, Should, Could, Won’t (или: Must, Should, Could, Would/Cancel/Optional, в някои варианти). Най-широко приетото наименование е MoSCoW analysis.

MoSCoW e метод за приоритизация.Използва се главно в управлението на проекти и разработката на продукти, но принципът е приложим към учебни програми, учебно съдържание и при планиране на учебни теми.

Четирите категории са:

* **Must** – абсолютно задължително, без което целта не може да бъде изпълнена
* **Should** – силно препоръчително, важно, но не критично (ако нещо трябва да отпадне, може да е това)
* **Could** – желателно, хубаво е да го има, но не е наложително
* **Won’t (или Would/Cancel)** – в дадения цикъл няма да се прави/учим, или отпада; може да се остави за бъдеще

Един от възможните варианти за прилагане на MoSCoW за анализ на учебна програма е:

1. Изберете критериите (например: какво трябва да знае всеки ученик до края на курса)
2. За всяка тема/раздел/урок определете:

* Това съдържание абсолютно необходимо ли е за базовата грамотност? (Must)
* Това съдържание би трябвало да го има, но може да се премести или свие, ако няма време? (Should)
* Това е хубаво да го имаме, но няма да е фатално, ако не остане време? (Could)
* Това няма да влезе в обучението сега, може да е свързано с други класове, профили, или отпада? (Won’t/Cancel)

Примерна таблица:

| Тема | Категория | Обосновка |
| --- | --- | --- |
| Основи на делението | Must | без това не може нататък |
| Деление с остатък | Should | важно, но може да се прехвърли |
| Странни делители | Could | интересно, но не е критично |
| Деление в бройна система X | Won’t/Cancel | извън програмата за този клас |

Този подход е полезен защото:

* Фокусирате се върху най-важното (особено при ограничено време или ресурси)
* Лесна адаптация според профила на учениците/училището
* Помага при внедряване на иновативни програми, където съдържанието трябва да се намали или промени

Официално този метод се използва:

* В образованието: при съставяне на авторски учебни програми, курсове, проектно-базирано обучение
* В ИТ системи, разработка на софтуер, бизнес анализ

Терминология:

* В образователен контекст, понякога на български се ползват термини като: Задължително, Важно, Пожелателно, Отпадащо
* На английски – MoSCoW (избягва се думата "Cancel", по-често се ползва "Won't")

**Контекст на приложимост:** Подходящ за приоритизация на учебното съдържание и подбор на най-съществените теми в учебната програма по програмиране, при ограничени ресурси и/или време.

## Използване на автентични инструменти и технологии в приложното програмиране

Технологиите постоянно се развиват и усъвършенстват, поради което учебният процес в областта на приложното програмиране трябва да се адаптира съобразно настоящите тенденции. Използването на автентични инструменти може да обогати учебния процес.

Прмер - интегриране на Python, PyCharm и други съвременни среди.

Python се превърна в един от най-популярните програмни езици благодарение на своята простота, гъвкавост и широк спектър от приложения. За учебни цели е препоръчително използването на PyCharm като интегрирана среда за разработка, тъй като тя позволява на учениците да се запознаят с професионалните практики в софтуерната индустрия. Може да се помисли и за **използване на Linux среди**. За потребители на Windows се препоръчва инсталирането на Linux среди (например Git Bash), което осигурява сходна работна среда с тази, използвана от професионалистите.

**Изтеглянето на учебни материали от GitHub**: Използването на Jupyter Notebooks за представяне на учебния материал дава възможност за интерактивно обучение и директно прилагане на наученото.

# Методология (технология)

Целта е да се изработят детайлизирани училищни учебни програми за предметите от професионалната подготовка по специалност „Приложно програмиране“. За да се гарантира максимална ефективност, приобщаване и автентичност, процесът на адаптиране се основава на интеграция на доказани педагогически модели и принципи.

**Акцентът върху „училищни“** подчертава нуждата от адаптиране на учебните програми според конкретните условия – профила и възможностите на учениците, квалификацията на преподавателите, спецификата на материалната база и традициите на училището. Това създава собствен облик и педагогическа „школа“, което позволява постигането на високи резултати в реален контекст.

**Детайлизирането** означава разбиване на всяка учебна програма на седмичен и дневен обхват, като за всяко занятие се изготвя конкретен план с теоретични, практически и оценъчни компоненти. Съществен елемент са разнообразните дидактически материали – теоретични текстове, примерни решения, задачи за групова и индивидуална работа, тестове и инструменти за проверка.

## Алгоритъм и ключови стъпки

Методологичният подход се реализира поетапно, като всяка стъпка е целенасочено обвързана с проверени образователни модели:

1. Анализ на рамковия учебен план (рамкова програма)

* От всички дисциплини се извеждат предметите от професионалната подготовка.
* Принципът FL/W (Fit/Lend/Optimize/Work) тук се прилага чрез съобразяване на учебния план със спецификата на училището и наличните ресурси (Fit), както и чрез предварително ангажиране на целия екип – за съдействие и взаимопомощ (Lend/Work).

1. Групиране по теория/практика и установяване на синхрон

* Всяка дисциплина се структурира на блокове по теория и практика, със стремеж за непрекъсната връзка между овладяване на концепции и приложна работа.
* Моделът „Постепенно освобождаване на отговорност“ (GRR) присъства именно тук: започва се с по-висока роля на учителя при въвеждане на новите теми (директно обучение), преминава се през ръководена практика, сътруднически задачи и се стига до самостоятелна продуктивност на учениците в практическите занятия.

1. Определяне на учебните седмици/разписание

* За всяка дисциплина се определя броят на учебните седмици, което позволява реалистично разпределяне на съдържанието.

1. Формулиране на цели и задачи по предмети

* За всеки предмет се формулират ясни цели и очаквани резултати, стъпвайки на изискванията на стандарта, но адаптирани към спецификата на училището.

1. MoSCoW анализ на съдържание

* За всяка програма – теория и практика – се извършва приоритизационен MoSCoW анализ, при който съдържанието и темите се разпределят по следните категории: Must (задължително ядро), Should (важно, но не критично), Could (пожелателно), Won’t (отпадащо).
* Това дава възможност за ефективно адаптиране и фокусиране върху ключовите теми, спрямо времето, ресурсите и целевия профил на учениците.

1. Разбиване на програмата на занятия (уроци)

* Всеки предмет се описва на ниво занятие/урок, като може да включва въвеждащи часове за нови знания, примери, упражнения, самостоятелна и групова работа, проверка и обобщение.
* 5PPPS моделът се прилага тук: всеки урок следва етапите Bridging-In (контекст и мотивация), Stated Outcomes (ясни цели), Pre-assessment (предварителна диагностика), Participant Learning (учене чрез действие), Post-assessment (оценка) и Summary (обобщение на наученото).

1. Синхронизиране на теория и практика

* Паралелно се поддържа връзка между теоретичните и практически занимания, така че факторите за учене чрез правене да са водещи – включително реални казуси от програмирането и софтуерната разработка.

1. Разработване на уроците

* За всяко занятие се съставя детайлен план с тайминг, дидактически материали, примери, задачи и тестове.
* GRR моделът тук подпомага планирането на прехода от учителска към ученическа активност, а принципът FL/W помага за оптимален учебен климат.

1. Итеративност и екипна работа

* Целият процес се осъществява в екип от преподаватели. Всеки етап преминава през обсъждания, корекции и обратна връзка, което осигурява високо качество и устойчивост на резултатите. Необходимо е въвеждане на итеративност – редовни цикли „разработка → обсъждане → корекция → повторна работа“.

## Откроени характеристики и предимства

### Интеграция на съвременни принципи:

Включването на модели като 5PPPS, GRR, MoSCoW и FL/W осигурява поливалентност и баланс между теоретичното и практическото обучение, между индивидуалното и груповото учене.

### Приобщаващ и адаптивен подход:

Отчитат се индивидуалните потребности на учениците, като адаптацията стъпва както на актуална диагностика (pre-assessment), така и на постоянна обратна връзка (post-assessment).

### Ефективна приоритизация:

Чрез MoSCoW анализа фокусът се задържа върху най-важните теми, без да се претоварват програмите с излишно или трудно приложимо съдържание.

### Осигуряване на автентичност и мотивация:

Чрез Bridging-In и въвеждане на реални казуси, учениците виждат смисъл и реална връзка с професионалната реалност.

### Обобщение

Предложеният алгоритъм предлага ясно структурирана, педагогически обоснована, подходяща за българското училище технология за детайлизиране и прилагане на училищни учебни програми по „Приложно програмиране“. Чрез активното използване на интегрирани съвременни методики се осигурява пълноценна и адаптивна подготовка за учениците в контекста на динамично развиващия се свят на дигиталните технологии.

# Реализация

## Стъпка 1: Анализ на рамковия учебен план

На тази стъпка изходният документ е рамковият (типовият) учебен план за специалността, който е утвърден от министъра на образованието и е публикуван на сайта на министерството. [1] Т.к ни интересуват само предметите от професионалната поготовка на учениците, изваждаме само радели от III до V от учебния план включително. Резултатът (по-точно началото) е показан в таблицата:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Специалност ПРИЛОЖНО ПРОГРАМИРАНЕ - *вариант В4 -* трета степен на професионална квалификация с разширено изучаване на чужд език - дневна форма на обучение | | | | | | | | | | | | |
|  | Учебни предмети/модули за професионална подготовка | I гимназиален етап | | | | | II гимназиален етап | | | | | ОБЩО |
| Класове | | | | Общо | Класове | | | | Общо |
| VІІІ | ІХ | Х | | VІІІ – Х | ХІ | | ХІІ | | XІ – ХII | VІІІ – ХІІ |
|  | Учебни седмици | 36 | 36 | 36 | 2 |  | 36 | 2 | 29 | 2 |  |  |
| *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* |
| Раздел Б - избираеми учебни часове | | | | | | | | | | | | |
| III. | Отраслова професионална подготовка |  |  |  |  |  |  |  |  |  |  |  |
| 1. | Теория на професията |  |  |  |  |  |  |  |  |  |  |  |
| 1.1. | Увод в програмирането |  | 18 |  |  | 18 |  |  |  |  |  | 18 |
| 1.2. | Програмиране |  | 36 |  |  | 36 |  |  |  |  |  | 36 |
| 1.3. | Увод в обектно-ориентираното програмиране |  | 18 |  |  | 18 |  |  |  |  |  | 18 |
| 1.4. | Увод в алгоритмите и структурите от данни |  |  | 36 |  | 36 |  |  |  |  |  | 36 |
| 1.5. | Обектно-ориентирано програмиране |  |  |  |  |  | 72 |  |  |  | 72 | 72 |
| 1.6. | Бази данни |  |  |  |  |  | 36 |  |  |  | 36 | 36 |
| 1.7. | Разработка на софтуер |  |  |  |  |  | 72 |  |  |  | 72 | 72 |

## Стъпка 2: Групиране по теория/практика и установяване на синхрон

Получената на предишната стъпка от алгоритъма таблица я преобразуваме по следния начин:

* Преподреждаме таблицата като всяка дисциплина се структурира на блокове по теория и практика.
* Добавяме колони „клас“, „тип“ и „годишен брой часове“. В първата от тези колони полълваме класа, във втората – типа на предмета (теория или учебна практика), а във третата пренасяме определиения в плана годишен хорариум за съответния предмет.

## Стъпка 3: Определяне на учебните седмици

За всяка дисциплина се определя броят на учебните седмици, през които тя ще се изучава:

* Добавяме колони „брой учебни седмици“ и „брой часове седмично“
* В колоната „брой учебни седмици“ (след обсъждане в екипа) определяме броят на учебните седмици, през които ще се изучава съответният учебен предмет. Ако предметът ще се изучава в рамките на един учебен срок – отбелязваме 18 седмици (11 за втори срок на 12-ти клас). Ако предметът ще се изучава в рамките на цялата учебна година – отбелязваме 36 седмици (29 за 12-ти клас).
* В колоната „брой часове седмично“ отбелязваме седмичния хорариум за учения предмет. Ако предметът ще се изчава в два срока с различен седмичен хорариум посочваме две числа ( за първи и за втори срок), разделени с наклонена черта. Например запис „2/3“ ще означава два часа през първия и три часа седмично през втория учебен срок. Седмичен хорариум изразен с едно число означава един блок със съответната продължителност. При необходимост от разделяне на блокове се използва синтаксис, наподобяващ аритметичен израз. Например, „2х1+1х2“ би означавало два пъти седмично по един час и веднъж седмично блок от два часа за седмичен хорариум от 4 часа.

Крайният резултат би трябвало да изглежда като показаният в таблицата:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Учебни предмети/модули за професионална подготовка | клас | тип | годишно | | |
| брой часове | брой учебни седмици | часове седмично |
| Увод в програмирането | 9 | Теория | 18 | 18 | 1 |
| Практика | 36 | 18 | 2 |
| Програмиране | 9 | Теория | 36 | 18 | 2 |
| Практика | 36 | 18 | 2 |
| Увод в обектно-ориентираното програмиране | 9 | Теория | 18 | 18 | 1 |
| Практика | 18 | 18 | 1 |
| Разширена професионална подготовка | 9 | Теория | 18 | 18 | 1 |
| Практика | 36 | 18 | 2 |
| Увод в алгоритмите и структурите от данни | 10 | Теория | 36 | 18 | 2 |
| Практика | 36 | 18 | 2 |
| Разширена професионална подготовка | 10 | Теория | 36 | 18 | 2 |
| Практика | 54 | 18 | 3 |
| Обектно-ориентирано програмиране | 11 | Теория | 72 | 36 | 2 |
| Практика | 72 | 36 | 2 |
| Бази данни | 11 | Теория | 36 | 18 | 2 |
| Практика | 36 | 18 | 2 |
| Разработка на софтуер | 11 | Теория | 72 | 36 | 2 |
| Практика | 90 | 36 | 2/3 |
| Програмиране за вградени системи | 11 | Теория | 36 | 18 | 2 |
| Практика | 36 | 18 | 2 |
| Математически основи на програмирането | 11 | Теория | 36 | 18 | 2 |
| Практика | 36 | 18 | 2 |
| Операционни системи | 11 | Теория | 36 | 18 | 2 |
| Практика | 36 | 18 | 2 |
| Конкурентно програмиране | 11 | Теория | 18 | 18 | 1 |
| Практика | 36 | 18 | 2 |
| Разширена професионална подготовка | 11 | Теория | 36 | 18 | 2 |
| Практика | 36 | 18 | 2 |
| Алгоритми и структури от данни | 12 | Теория | 58 | 29 | 2 |
| Практика | 87 | 29 | 3 |
| Функционално програмиране | 12 | Теория | 29 | 29 | 1 |
| Практика | 29 | 29 | 1 |
| Интернет програмиране | 12 | Теория | 58 | 29 | 2 |
| Практика | 116 | 29 | 4 |
| Софтуерно инженерство | 12 | Теория | 58 | 29 | 2 |
| Практика | 87 | 29 | 3 |
| Разширена професионална подготовка | 12 | Теория | 29 | 29 | 1 |
| Общо |  |  | 1649 |  | 68/69 |

## Стъпка 4: Формулиране на цели и задачи по предмети

За всеки предмет се формулират ясни цели и очаквани резултати, стъпвайки на изискванията на стандарта, но адаптирани към спецификата на училището.

## Стъпка 5: MoSCoW анализ на съдържание

За всяка програма – теория и практика – се извършва приоритизационен MoSCoW анализ, при който съдържанието и темите се разпределят по следните категории: Must (задължително ядро), Should (важно, но не критично), Could (пожелателно), Won’t (отпадащо).

Важно е преди да се пристъпи към анализ на конкретните учебни програми екипът да разработи и приеме Критерии за категоризация.

Аз лияно, базирайки се на своя професионален опит като педагог и програмист препоръчвам следните базови критерии:

**Must (Задължителни теми):**

* Теми, без които ученикът не може да разбере базовите принципи на обектно-ориентираното програмиране.
* Абсолютно необходими за покриване на държавния образователен стандарт (ДОС) и за основна грамотност по предмета.
* Без овладяването им по-нататъшното учене/развитие е невъзможно.

**Should (Важно, но не критично):**

* Теми, които са важни, силно препоръчителни за добра подготовка, но могат да се съкратят/изместят при ограничени ресурси или време.
* Дават допълнителна стойност, задълбочаване или практическа насоченост.

**Could (Пожелателно):**

* Хубаво е да се преподават, развиват интерес или по-абстрактни идеи, които не са фундаментално изискване.
* Може да бъдат включени ако има време, интерес, мотивирана група или училището има ресурси.

**Won’t/Cancel (Не влиза, Отпада):**

* Теми, които са неприложими, остарели, твърде сложни или не отговарят на възможностите на конкретния клас/училище.
* По решение на екипа те не се разглеждат изобщо в този учебен цикъл.

Нека разгледаме конкретен пример – програмата по Обектно-ориентирано програмиране за 11-ти клас, теория.

Програмата, утвърдена от МО е:

**Раздел 1. Дефиниране на класове за напреднали**

1. Дефиниране на класове
2. Полета и свойства
3. Методи
4. Методи и конструктори
5. Статични методи и конструктори
6. Статични полета и свойства
7. Памет, стек, хип. Разположение на обектите в паметта
8. Финализации и/или деструктори

**Раздел 2. Шаблонни класове**

1. Въведение в шаблонните класове
2. Шаблонни методи и интерфейси
3. Ограничители за шаблонни класове

**Раздел 3. Наследяване, абстракция, интерфейси**

1. Наследяване
2. Преизползване на класовете
3. Абстракция
4. Интерфейси

**Раздел 4. Полиморфизъм**

1. Полиморфизъм
2. Презареждане и презаписване
3. Абстрактни класове и полиморфизъм
4. Полиморфизъм чрез интерфейси

**Раздел 5. Работа с обекти**

1. Итератори
2. Компаратори
3. Отражение на типовете

**Раздел 6. Елементи от функционалното програмиране**

1. Ламбда изрази и функции
2. Ламбда функции и LINQ
3. Функции на LINQ за работа с колекции
4. Делегати и функционално програмиране

**Раздел 7. Комуникация между обекти. Събития и интерфейси.**

1. Комуникация между обекти. Въведение в събитийното програмиране. Делегати
2. Комуникация между обекти. Събития
3. Комуникацията между обекти. Аргументи на събития
4. Комуникация между обекти. Слушатели за събитие

**Раздел 8. Изключения**

1. Прихващане на изключения
2. Хвърляне на изключения

**Раздел 9. Работа с потоци и файлове**

1. Потоци
2. Стандартни потоци

**Раздел 10. Базови шаблони за дизайн**

1. Шаблони в проектирането при създаване

След MoSCoW анализа би се получил резултат, подобен на показания в таблицата:

| Тема | Категория | Обосновка |
| --- | --- | --- |
| Дефиниране на класове | Must | Базова ООП концепция |
| Полета и свойства | Must | Базова ООП концепция |
| Методи | Must | Базова ООП концепция |
| Методи и конструктори | Must | Задължително за правилна инициализация |
| Статични методи и конструктори | Should | Добре е да се знае, не винаги е критично |
| Статични полета и свойства | Should | Допълва концепции, роля при оптимизация |
| Памет, стек, хип. Разположение на обектите | Could | Полезно, дава основа за напреднали |
| Финализации и/или деструктори | Could | При по-нататъшни езици/платформи може да отпадне |
| Въведение в шаблонните класове | Should | Приложимо за абстракция, типова независимост |
| Шаблонни методи и интерфейси | Should | Задълбочено, но не базово |
| Ограничители за шаблонни класове | Won’t | Тясна употреба |
| Наследяване | Must | Ядро на ООП |
| Преизползване на класовете | Should | Практическо умение |
| Абстракция | Must | Ядро на ООП |
| Интерфейси | Must | Ядро на ООП, много важно за индустрията |
| Полиморфизъм | Must | Ядро на ООП |
| Презареждане и презаписване | Must | Основен принцип в ООП |
| Абстрактни класове и полиморфизъм | Should | Дава по-голяма дълбочина |
| Полиморфизъм чрез интерфейси | Should | Дава по-голяма дълбочина |
| Итератори | Should | Практическо, но не критично |
| Компаратори | Should | Практическо, но не критично |
| Отражение на типовете | Could | За напреднали, рядко се включва в начални курсове |
| Ламбда изрази и функции | Should | Актуална тенденция, препоръчително |
| Ламбда функции и LINQ | Won’t | По-тясна насоченост, свързано с конкретни езици |
| Функции на LINQ за работа с колекции | Could | По-тясна насоченост, свързано с конкретни езици |
| Делегати и функционално програмиране | Could | Напреднало, добре за интересуващите се |
| Комуникация между обекти. Въведение в събитийното програмиране. Делегати | Should | Полезно за реални системи |
| Комуникация между обекти. Събития | Should | Полезно за реални системи |
| Комуникацията между обекти. Аргументи на събития | Could | Детайл, при наличие на време/напредналост |
| Комуникация между обекти. Слушатели за събитие | Could | Детайл, при наличие на време/напредналост |
| Прихващане на изключения | Must | Задължително за писане на надежден код |
| Хвърляне на изключения | Must | Задължително за писане на надежден код |
| Потоци | Should | Полезно за файлови операции |
| Стандартни потоци | Could | Детайл, не е критично в началния курс |
| Шаблони в проектирането при създаване | Should | Практично, препоръчително |

## Стъпка 6: Разбиване на програмата на занятия (уроци)

Всеки предмет се описва на ниво занятие/урок, като може да включва въвеждащи часове за нови знания, примери, упражнения, самостоятелна и групова работа, проверка и обобщение. Броят на занятията, отделени на дадена тема е функция от резултатите на анализа в предишната точка и определеният в утвърдената от МО учебна програма минимален брой учебни часове по раздели. В конкретният случай той е:

|  |  |  |
| --- | --- | --- |
| № | Наименование на разделите | Минимален брой часове теория |
| 1. | Дефиниране на класове за напреднали | 6 |
| 2. | Шаблонни класове | 6 |
| 3. | Наследяване, абстракция, интерфейси | 6 |
| 4. | Полиморфизъм | 6 |
| 5. | Работа с обекти | 6 |
| 6. | Елементи от функционалното програмиране | 6 |
| 7. | Комуникация между обекти. Събития и интерфейси. | 6 |
| 8. | Изключения | 6 |
| 9. | Работа с потоци и файлове | 6 |
| 10. | Базови шаблони за дизайн | 6 |
|  | Общ минимален брой часове | 60 |
|  | Резерв часове | 12 |
|  | Общ брой часове | 72 |

Ето едно балансирано разпределение на темите от Раздел 1 по занятия, отчитайки резултата от MoSCoW анализа и ограниченията (3 занятия по 2 учебни часа, с опция за 4-то при необходимост). Стремежът е във всяко занятие да се намери място за Must темите, а Should/Сould да бъдат разпределени така, че времето да стигне и учениците да имат възможност за усвояване и прилагане.

Занятие 1

* Дефиниране на класове (Must)  
  Въведение и ключови понятия, примери
* Полета и свойства (Must)  
  Видове полета и свойства, достъп
* Методи (Must)  
  Създаване и използване, примери

*(Фокус: Изграждане на базова структура на клас. Практическа задача: Дефиниране на клас с основни полета, свойства и методи.)*

Занятие 2

* Методи и конструктори (Must)  
  Създаване и цел на конструктори, видове конструктори
* Статични методи и конструктори (Should)  
  Разлика между статични и инстанционни, обща роля
* Статични полета и свойства (Should)  
  Защо и кога се използват

*(Фокус: Инициализация на обекти, споделени характеристики на клас. Практическа задача: Създаване на клас с конструктори и статични елементи.)*

Занятие 3

* Памет, стек, хип. Разположение на обектите (Could)  
  Въведение в концепциите, демонстрация
* Финализации и/или деструктори (Could)  
  Роля на деструкторите, ситуации за използване

*(Фокус: Какво се случва „зад кулисите“ – основно за любопитните, обогатяващо знание. Практическа задача: Демонстрации и анализ на поведение при създаване и унищожаване на обекти.)*

Занятие 4 (резервно, при необходимост)

* Преговор и разширение по any Should/Could теми, които не са покрити в достатъчна дълбочина според темпото на групата.
* Допълнителни упражнения по конструктори, статични елементи или управление на памет.

Обобщение:

* Ядрото (Must) се разгръща още в първите две занятия.
* Всички теми Should са третирани до края на второто занятие, но могат да се разширят при допълнително време.
* Could темите са отделени за последното или резервното занимание, с идея да бъдат покрити, ако класът има интерес или напредва с добър темп.
* Всяко занятие комбинира теория и практика – минимална теория, след това веднага упражнения с код.

## Стъпка 7: Синхронизиране на теория и практика

Паралелно се поддържа връзка между теоретичните и практически занимания, така че факторите за учене чрез правене да са водещи – включително реални казуси от програмирането и софтуерната разработка. Основната идея е за всяка учебна дисциплина да се работи паралелно върху учебните програми по теория и по практика като се тъси максимален синхрон между двете. Практиката не може да изпреварва теорията. Практиката трябва да покрива изцяло теорията.

## Стъпка 8: Разработване на уроците

За всяко занятие се съставя детайлен план с тайминг, дидактически материали, примери, задачи и/или тестове.

Ето един примерен план на урок за Занятие 1 от предишната стъпка:

Детайлен план на **Занятие** 1 (90 минути)

****Тема:**** Дефиниране на класове, полета, свойства и методи

0. Подготовка (5 мин. преди започване****)****

* Проверка на техника, подготвяне на примерния код, отваряне на средата за програмиране.

****1. Въведение и мотивиране (Bridging-In) – 5 мин.****

* Мотивационен въпрос („Как бихте описали автомобил като обект?“).
* Кратък разговор/примери за „обекти“, които ученикът среща ежедневно.
* Преговор – обясняване връзката между реалния свят и класове в програмирането.

****2. Излагане на основните понятия (Stated Outcomes) – 5 мин.****

* Ясно заявяване на целите:
  + Да могат да дефинират клас.
  + Да създават и използват полета, свойства и методи.
  + Да разграничават основните части на класа.

****3. Изложение и примери (Обяснение + Демо код) – 15 мин.****

* **3.1. Дефиниране на клас**
  + Синтаксис, основни правила.
  + Пример клас: Car, Student, или друг лесно разбираем пример.
  + Практическа демонстрация – показване на кода в реално време.
* **3.2. Полета и свойства**
  + Дефиниция на полета (член-променливи).
  + Дефиниция на свойства (getters/setters, ако езикът ги поддържа).
  + Разлика между публично и частно поле.
* **3.3. Методи**
  + Какво е метод, как се създава и извиква.
  + Примери на прости методи – напр. PrintInfo(), IncreaseSpeed().

****4. Кратка дискусия и рефлексия – 7 мин.****

* Въпроси към класа: „Може ли да дадете друг пример за обект?“
* Критични въпроси: „Защо свойствата са по-добри от директен достъп до полетата?“
* Обратна връзка дали на всички е ясно.

****5. Упражнение 1 (Работа в клас – съвместно) – 8 мин.****

* Заедно с учениците дефинирате клас (например Book) с няколко полета и методи.
* Въвеждане на кода, моментално обяснение на всеки ред.

****6. Самостоятелна практика (Практическа задача) – 25 мин.****

* **Задача:**  
  „Създайте клас Animal с полета (например name, age), свойства за всяко поле, и поне два метода (напр. CelebrateBirthday, Speak).“
* Упътване: Дава се шаблон/пример, ако някой има нужда.
* Учителят обикаля и подпомага индивидуално.

****7. Проверка и споделяне на резултати – 10 мин.****

* Доброволци представят своето решение на дъската/проектора.
* Дискусия: Дали всички са използвали свойства? Кой метод изглежда по-оригинално?

****8. Кратък тест/мини-викторина (Pre/Post Assessment) – 7 мин.****

* 3 бързи въпроса (на хартия или дигитално), например:
  + Как се дефинира клас?
  + Каква е разликата между поле и свойство?
  + Как се извиква метод?

****9. Обобщение и Самостоятелан работа (Summary & Homework) – 8 мин.****

* Обобщаване на ключовите моменти:
  + Клас = шаблон, полета = данни, свойства = контрол, методи = действия.
* Домашна задача:
  + Подобно на упражнението в клас – направи клас Person с уникално име, възраст, и няколко метода по избор.

****Общ график (Тайминг)****

| Част | Време (мин) |
| --- | --- |
| Въведение, мотивация | 5 |
| Излагане на цели | 5 |
| Обяснение и примери | 15 |
| Дискусия/рефлексия | 7 |
| Съвместно упражнение | 8 |
| Самостоятелна практика | 25 |
| Представяне/проверка | 10 |
| Мини-викторина | 7 |
| Обобщение и домашно | 8 |
| **Общо** | **90** |

Към така формирания детайлен план на занятието е добре да се присъединят **кратки теоретични бележки, примерен код и подходящи задачи** за Занятие 1:

Теоретични бележки и примери

1. ****Дефиниране на класове****

* В Python клас се дефинира с ключовата дума class:

# Пример за клас

class Car:

pass

* Класът е шаблон за създаване на обекти.

2. ****Полета (атрибути) и свойства****

* Полетата са променливи, които принадлежат към обекта (инстанцията):

class Car:

def \_\_init\_\_(self, brand, year):

self.brand = brand # поле (атрибут)

self.year = year # поле (атрибут)

* Свойства (property) позволяват контрол на достъпа до полетата:

class Car:

def \_\_init\_\_(self, brand, year):

self.\_year = year

@property

def year(self):

return self.\_year

@year.setter

def year(self, value):

if value > 1885: # първата кола

self.\_year = value

else:

print("Грешна година!")

В практиката, често се използват директно атрибути, но свойствата се създават при нужда от контрол и проверка!

3. ****Методи****

* Методите са функции, които живеят вътре в класа и обикновено работят с атрибутите чрез self:

class Car:

def \_\_init\_\_(self, brand):

self.brand = brand

def print\_info(self):

print(f"Това е автомобил {self.brand}")

Примерен цялостен клас

class Animal:

def \_\_init\_\_(self, name, age):

self.name = name # поле

self.age = age # поле

def speak(self):

print(f"{self.name} казва: Мяу!" )

def celebrate\_birthday(self):

self.age += 1

print(f"Честит рожден ден, {self.name}! Вече си на {self.age} години.")

Практически задачи и упражнения

****Съвместно (с учителя):****

Създайте клас Book с полета title и author. Добавете метод print\_info, който показва информация за книгата.

**Решение:**

class Book:

def \_\_init\_\_(self, title, author):

self.title = title

self.author = author

def print\_info(self):

print(f"\"{self.title}\" написана от {self.author}.")

****Самостоятелна работа (за учениците):****

1. Създайте клас Animal с полета name и age.
2. Добавете свойства (property) за двата атрибута.
3. Добавете два метода: speak (който да извежда <име> казва нещо по избор) и celebrate\_birthday (който увеличава годините с 1 и извежда поздравление).

Примерен резултат:

class Animal:

def \_\_init\_\_(self, name, age):

self.\_name = name

self.\_age = age

@property

def name(self):

return self.\_name

@property

def age(self):

return self.\_age

def speak(self):

print(f"{self.name} казва: Гррр!")

def celebrate\_birthday(self):

self.\_age += 1

print(f"Честит рожден ден, {self.name}!")

Мини-викторина (бърз тест)

1. Как се дефинира клас в Python?
2. Кое е разликата между поле и метод?
3. Защо е полезно да използваме свойства (property)?
4. Как се създава обект от класа Book?

Примери за верни отговори:

1. С ключовата дума class.
2. Полето пази данни, методът прави действие или връща стойност.
3. За да контролираме четенето/писането, напр. валидация.
4. b = Book("Заглавие", "Автор")

Самостоятелна работа (пример)

1. Създайте клас Person с полета name и age.
2. Добавете метод say\_hello, който извежда поздравление.
3. Добавете метод have\_birthday, който увеличава възрастта с 1.

За всяко занятие е добре да имаме готовност да подкрепим разясненията с още живи примери и ако някой ученик работи по-бързо – да го насърчим за самостоятелни експерименти (например метод, който връща описание на животното).

# Заключение

В заключение, установено е, че използването на методологии като BOPPPS, GRR, MSCW и принципа FLOW, заедно с интеграция на автентични технологични инструменти, представлява най-подходящата стратегия за адаптиране на рамковия учебен план по „Приложно програмиране“. Тези методи, подкрепени от опита на международни и национални образователни инициативи, осигуряват необходимата гъвкавост, качество и ефективност, които са от съществено значение за подготовката на учениците за бъдещите предизвикателства в динамичната софтуерна индустрия.

Настоящият анализ дава изчерпателен алгоритъм за адаптиране, който може да бъде интегриран в съществуващите учебни процеси, като по този начин се подобрява качеството на образованието и се гарантира, че бъдещите специалисти ще бъдат добре подготвени за реалния свят на информационните технологии.

Предложеният модел за съставяне на училищни учебни програми по специалността „Приложно програмиране“ е насочен към постигане на баланс между съвременни педагогически принципи и реалните нужди на учениците и преподавателите. Методологичният подход позволява детайлно, ефективно и адаптивно планиране, което насърчава развитието на практически умения, креативност и мотивираност.

Осъзнавам, че динамиката на технологичната среда и разнообразието в ученическите групи изискват непрекъснато усъвършенстване на учебния процес. Затова поканвам колегите учители, експерти и всички заинтересовани страни да споделят опит, предложения и идеи за надграждане на настоящата методика. Вярваме, че чрез сътрудничество и обратна връзка ще постигнем устойчиво подобряване на качеството на професионалното образование в областта на приложното програмиране.

# Източници

1. https://www.mon.bg/obshto-obrazovanie/uchebni-planove-i-programi-2/
2. https://lex.bg/en/laws/ldoc/2136914662
3. https://journals.plos.org/ploscompbiol
4. https://www.ontario.ca/page/children-special-needs
5. https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1009184
6. https://softuni.foundation/projects/applied-software-developer-profession/
7. https://specialeducationnotes.co.in/HIC13UNIT4.htm
8. https://tlconestoga.ca/active-learning-in-your-tech-or-software-lessons/
9. <https://mindhub.bg/>
10. <https://softuni.bg/>
11. https://prepodavame.bg/adaptirane-na-uchebno-sadarzhanie-za-uchenitsi-sas-sop-zashto-koga-i-kak/