Завдання, підібрані вчителем для формування понять, повинні мати системний характер щодо засвоєння кожного окремого поняття у взаємозв'язку з іншими поняттями.

**БІБЛІОГРАФІЯ**

1. Выготский Л. С. Собрание сочинений : В 6 т. / Л. С. Выготский. – М. : Педагогика, 1982. – Т.1. – 487 с. – Т.2. – 504 с.

2. Концепція Державної цільової соціальної програми підвищення якості шкільної природничо-математичної освіти на період до 2015 року (схвалено розпорядженням Кабінету Міністрів України від 27 серпня 2010 р. № 1720-р.) [Електронний ресурс]. – Режим доступу: http://zakon4.rada.gov.ua/laws/show/1720-2010-%D1%80.

3. Корольова В. М. Актуальні проблеми розвитку природничо-математичної освіти в сучасній школі / В. М. Корольова [Електронний ресурс]. – Режим доступу: http://virtkafedra.ucoz.ua/el_gurnal/pages/vyp14/Koroljova.pdf.

4. Матяш О., Прус А. Окремі аспекти формування математичних понять / О. Матяш, А. Прус // Вісник Житомирського державного університету. – 2010. – Випуск 53. – С. 87-93 [Електронний ресурс]. – Режим досупу: http://eprints.zu.edu.ua/4580/1/vip_53_17.pdf.

5. Сверчевська І. А. Методична система вивчення геометричних тіл у загальноосвітній школі: Автореф. дис... канд. пед. наук: 13.00.02 / І. А. Сверчевська ; Нац. пед. ун-т ім. М.П.Драгоманова. – К., 2007. – 20 с.

6. Слєпкань З.І. Методика навчання математики : [підруч. для студ. мат. спеціальностей пед. навч. закладів] / З. І. Слєпкань. – К. : Зодіак-ЕКО, 2000. – 512 с.

7. Тарасенкова Н. А. Використання знаково-символічних засобів у навчанні математики : [монографія] / Н. А. Тарасенкова. – Черкаси : Відлуння-Плюс, 2002. – 400 с.

**ВІДОМОСТІ ПРО АВТОРА**

**Голодюк Лариса Степанівна** – заступник директора з науково-методичної діяльності комунального закладу «Кіровоградський обласний інститут післядипломної педагогічної освіти імені Василя Сухомлинського», кандидат педагогічних наук, доцент.

*Коло наукових інтересів:* теорія та методика навчання математики.

# МЕТОДИ Й ЗАСОБИ ВІЗУАЛЬНОГО ПРОЕКТУВАННЯ СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

***Валерій ГРИЦЕНКО***

*Описано засоби графічного моделювання за допомогою UML. Розглянуто процес розробки програмного забезпечення на етапі його проектування за допомогою інструментів графічного моделювання відповідно до об'єктно-орієнтованого підходу. Наведено приклад використання UML інструментарію.*

*The tools of graphical modeling with UML are described. The software development by using a graphical modeling according to the object-oriented approach is considered. An example of UML tools using is shown.*

**Постановка проблеми.** Через поступове зростання складності програмних систем, а від так і їх розробки, ще у 80-х роках минулого століття почали з'являтися спеціальні методи і засоби проектування програмного забезпечення, так звані CASE (від англ. Computer Aided Software Engineering). Сучасні CASE-засоби, на відміну від початкових, які обмежувалися лише завданнями автоматизації розробки програмного забезпечення, охоплюють широкі сфери технологій проектування інформаційних систем: від простих засобів аналізу і документування до повномасштабних засобів автоматизації, що охоплюють увесь життєвий цикл програмного забезпечення.

Найбільш трудомісткими етапами розробки інформаційних систем є етапи аналізу і проектування, у процесі яких CASE-засоби забезпечують якість прийнятих

технічних рішень та підготовку проектної документації. При цьому велику роль відіграють методи візуального представлення інформації. Це передбачає побудову структурних чи інших діаграм, використання різноманітної кольорової палітри, наскрізну перевірку синтаксичних правил. Графічні засоби моделювання предметної галузі дозволяють розробникам наочно бачити та вивчати існуючу інформаційну систему, перебудовувати її відповідно до поставлених цілей і наявних обмежень.

**Метою статті** є дослідження можливостей засобів візуального проектування, а також розгляд основних етапів проектування программного забезпечення на їх основі.

Нині на ринку інформаційних технологій налічується низка програмних продуктів CASE категорії, які підтримують існуючі підходи до розробки програмного забезпечення.

До підходів проектування програмного забезпечення можна віднести такі:

• структурний, в основу якого покладено принцип алгоритмічної декомпозиції – структура системи описується термінами ієрархії її функцій і передачі інформації між окремими функціональними елементами (модулями);

• об'єктно-орієнтований, який використовує об'єктну декомпозицію – структура системи визначається множиною об'єктів і зв'язків між ними, а поведінка системи описується термінами обміну повідомленнями між об'єктами.

Структурний підхід має більш тривалу історію в порівнянні з об'єктно орієнтованим, тому CASE-засоби спочатку підтримували методи структурного підходу, наприклад методологію функціонального моделювання і графічного опису процесів (IDEF0, IDEF1, IDEF3), побудову діаграм потоків даних (DFD). Найпопулярнішими у користувачів CASE- засобами можна назвати такі інформаційні системи, як All Fusion Process Modeller, ARIS, Ramus та інші.

Завдяки активізації розвитку об'єктно-орієнтованого підходу з'являється все більше програмних засобів, що реалізують методи об'єктно-орієнтованого опису інформаційних систем, наприклад Rational Rose, Visual Paradigm UML, UModel, Umbrello, Enterprise Architect та багато інших. До того ж, підтримку методів об'єктно-орієнтованого підходу забезпечують і нові версії деяких CASE-систем, що традиційно вважалися структурними, зокрема система ARIS [1].

Основним методом для опису інформаційної системи відповідно до об'єктно-орієнтованого підходу є універсальна мова моделювання UML (від англ. Unified Modeling Language), розроблена в 90-х рр. Г. Бучем, Д. Рамбо та І. Якобсоном.

У сучасній версії мови UML виділяють близько 15 видів діаграм для опису структури і поведінки проектованого програмного забезпечення [2]. Однак через різноманіття видів UML-діаграм вважаємо за доцільне визначити типову послідовність дій щодо застосування цього інструменту на етапі проектування програмного забезпечення.

**Етапи проектування програмного забезпечення**

Перше, що потрібно зробити перед проектуванням – це *визначити користувачів* майбутньої системи і, орієнтуючись на їхні побажання, сформулювати основні вимоги до програмного продукту. Тобто спочатку потрібно дізнатися, які дії користувачі хочуть автоматизувати (перекласти на систему). Потім, поступово обмежуючи або, навпаки, розширюючи, коло функціональності і користувачів можна уточнювати межі системи. Отже, знаючи зовнішніх, по відношенню до системи, виконавців і вимоги до неї, ми можемо отримати більш чітку межу проекту.

Доступна користувачам функціональність проекту зазвичай описується в документі з назвою «*Опис вимог*»[3]. Це потрібно для того, щоб між клієнтами і

розробниками було досягнуто розуміння з питання про те, що система повинна робити і чого не повинна.

Підготувавши всю цю документацію, можна розпочинати наступний етап – проектування діаграм *прецедентів*. Кожен прецедент має власне завдання, яке він повинен виконати, а їх набір встановлює всі можливі шляхи використання системи.

Особливу увагу на цьому етапі слід приділяти текстовому опису, в якому вказується реакція системи на певні фактори середовища (що станеться за певних умов).

Наступним етапом проектування програмного забезпечення є визначення його архітектури та основних складових. Цей етап проектування може забезпечити UML-діаграма *компонентів*.

Далі відповідно до об'єктно-орієнтованого підходу, необхідно виділити типи розглядуваних сутностей, їх характеристики (атрибути) і операції, які над ними можуть виконуватись (методи). В UML це можна зробити за допомогою побудови діаграми *класів*, на якій можна показати не лише типи об'єктів у вигляді класів, а й різні зв'язки між ними.

На наступному етапі проектування інформаційної системи слід визначити життєвий цикл кожного з перелічених раніше класів, тобто скласти перелік станів і переходів між ними. Для цього в UML передбачено побудову діаграм *станів*, які ґрунтуються на використанні апарату дискретної логіки і кінцевих автоматів.

Після визначення станів життєвого циклу всіх наявних класів доцільно відобразити взаємодії між їх об'єктами, які знаходяться в різних станах. Тут допомогти може побудова UML-діаграми *послідовностей*, яка надає можливість показати не лише об'єкти різних класів в різних станах, але й обмін повідомленнями між ними.

Однак крім технічних аспектів проектування і реалізації програмного забезпечення варто також описати технологію роботи з ним, тобто процеси розробки та експлуатації. З цією метою в UML можна скористатись засобами діаграми *діяльності*, яка дозволяє описати логічну послідовність дій та їх виконавців, а також показати пов'язані з процесами об'єкти. Отже, UML-діаграми діяльності детально відображають алгоритми виконання процесів.

**Огляд сучасних CASE-засобів. У** проведеному нами дослідженні проаналізовано низку засобів проектування інформаційних систем (Таблиця 1). До аналізу було обрано наступні характеристики систем: платформа, на якій може працювати система; наявність відкритого коду; чи наявна можливість генерування коду системи мови якими можна генерувати такий код; чи забезпечується функціонал оберненого інжинірингу, і якщо так, то якою мово; чи наявна підтримка стандарту обміну ме́та-інформацією (XMI).

На основі проведеного аналізу [4-12] можна виокремити найбільш вдалу і широко вживану на сьогодні систему Visual Paradigm for UML. Це система управління вимогами, яка підтримує повний цикл розробки програмного продукту – аналіз, дизайн архітектури, розробку програмного коду, тестування і розміщення продукту на стороні замовника. Visual Paradigm також забезпечує підтримку версійності і одночасної роботи команди користувачів над одним проектом, зокрема дозволяє різним членам команди працювати над однією діаграмою і порівнювати зміни, внесені користувачами на різних етапах проекту.

Моделювання діаграм в системі реалізовано на досить високому інтуїтивно зрозумілому рівні.

Таблиця 1

Порівняння CASE-засобів проектування

| Назва | Виробник | Платформа | Відкритий код | Мова генерування коду | Мова оберненого інжинірингу | Підтримка XMI |
|---|---|---|---|---|---|---|
| Erwin | Erwin | Windows | ні | - | - | ні |
| MS Visio | Microsoft | Windows | ні | - | - | ні |
| Dia | Alexander Larsson/GNOME Office | GTK+ (cross-platform) | так | - | - | ні |
| Enterprise Architect | Sparx Systems | Windows | ні | C++, Java, C#, VB, VB.Net, Visual Basic, Delphi, PHP, Python | C++, Java, C#, VB, VB.Net, Visual Basic, Delphi, PHP, Python | так (XMI 1.0, 1.1, 1.2) |
| Umbrello | Umbrello Team | Unix, Linux | так | C++, Java, C#, PHP, JavaScript, ActionScript, SQL, Pascal, Ada, Python, IDL, XML Schema, Perl, Ruby | C++, IDL, Pascal/Delphi, Ada, Python, Java, Perl | так |
| ArgoUML | tigris.org | Java (cross-platform | так | - | - | так (XMI 1.1, 1.2) |
| UModel | Altova | Windows | ні | Java 1.4, Java 5.0, Java 6.0, C# 1.2, C# 2.0, C# 3.0, VB 7.1, VB8.0 и VB 9.0 | C#, VB.NET и Java | так (XMI 2.1) |
| Rational Rose | Rational Software Corporation | Windows, Sun SPARC stations (UNIX, Solaris, SunOS), Hewlett-Packard (HP UX), IBM RS/6000 | ні | C++, Smalltalk, PowerBuilder, Ada, SQLWindows и ObjectPro | C++, Smalltalk, PowerBuilder, Ada, SQLWindows и ObjectPro | так |
| Visual Paradigm | Visual Paradigm | Linux, Mac OS X, Windows | ні | C#, VB .NET, Object Definition Language (ODL), Flash ActionScript, Delphi, Perl, Objective-C, Ruby | Java, C++, CORBA IDL, PHP, XML Schema, Ada, Python ,Java class, .NET dll u exe, JDBC | так (XMI 1.0, 1.2, 2.1) |

До особливих переваг Visual Paradigm у порівнянні з іншими системами слід віднести також широкі можливості для створення моделей, так зокрема аналітикові в межах одного проекту надається можливість опису вимог, з одночасним використанням стандартів і нотації моделювання: UML, BMPN, OMG, Mind Maps і ArchiMate.

**Приклад створення діаграми прецедентів у системі Visual Paradigm**

Як приклад наведемо процес створення діаграми прецедентів автоматизованої інформаційної системи управління ВНЗ модуль «Управління інформацією про співробітника».

Для заповнення діаграми потрібно розмістити на ній акторів та варіанти використання, користуючись блоками Actor Grid та Use Case Grid.

Для нашої предметної області вводимо наступних акторів:

Таблиця 2

| Актор | Стислий опис |
|---|---|
| Фахівець по роботі зі співробітниками | Співробітник, який безпосередньо спілкується зі співробітником та працює з даними про нього |
| Фахівець з організації нормативно-правової діяльності | Співробітник, який слідкує за виходом та реалізацією наказів |

Опишемо, які можливості повинна надавати система:

- актор «Фахівець по роботі зі співробітниками» використовує систему для створення, редагування даних про роботу співробітника та управління особистою інформацією про співробітника навчального закладу;

- актор «Фахівець з організації нормативно-правової діяльності» використовує систему для внесення, видалення та перегляду існуючих наказів. Зміна наказів може відбуватися до проведення наказів в базі.

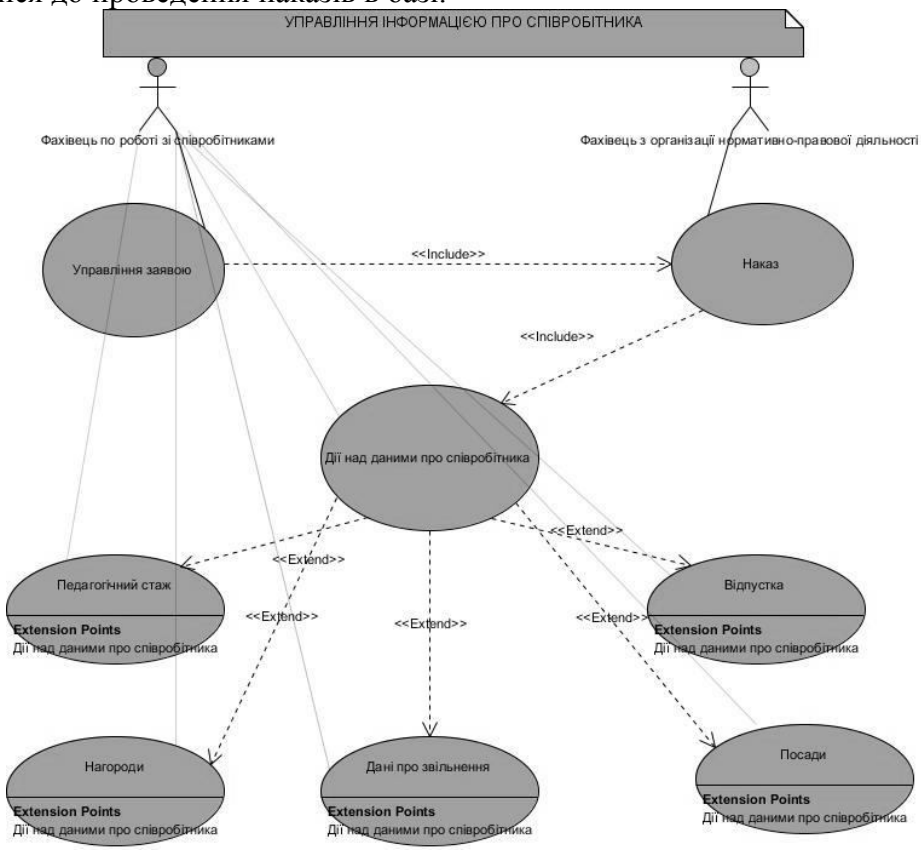


Рис.1. Діаграма прецедентів

На основі описаних можливостей визначаємо прецеденти:

Таблиця 3

Опис прецедентів та їх характеристик

| Прецедент | Стислий опис |
|---|---|
| Дії над даними про співробітника | Запускається фахівцем по роботі зі співробітниками. Дозволяє вносити, змінювати та переглядати дані про співробітника на основі наказів |
| Заява | Дозволяє додавати, змінювати чи видаляти заяву про прийняття на роботу |
| Дані про звільнення | Інформація про причину звільнення |
| Посади | Список можливих посад, які займають співробітники |
| Наказ | Запускається фахівцем з організації нормативно-правової діяльності. Дозволяє вносити, змінювати (поки вони не проведені), видаляти та переглядати існуючі накази. Також потрібно передбачити довідник по наказах |
| Відпустка | Опис кількості фактичних вихідних днів, які надані співробітнику |
| Нагороди | Облік нагород, премій, відзнак і т.д. |
| Педагогічний стаж | Нотування стороннього педагогічного стажу |

Розглянемо відношення між акторами та прецедентами. На мові UML можливий лише один тип відношення між актором та прецедентом – *відношення комунікації*. Тому всіх акторів ми зв'язуємо з прецедентами відношенням *Association.* Оскільки інший тип відношень тут задати не можна, то стереотип *communicate* можна не вказувати.

Відношення між всіма прецедентами, крім Заява, Наказ та Дії над даними про співробітника – *відношення розширення*, оскільки коли актор *Фахівець по роботі зі співробітниками* працює з даними про співробітника (оформляє, змінює і т.д.), то при цьому він не управляє інформацією про накази, яка зв'язана із замовленням.

Відношення між прецедентами Заява, Наказ та Дії над даними про співробітника – відношення включення, оскільки для створення нового співробітника обов'язково потрібно створити заяву на прийняття на роботу і відповідний наказ.

*Потік подій для прецедентів будемо описувати за наступним шаблоном:*

- X.1 передумови;
- X.2 головний потік;
- X.3 підпотоки;
- X.4 альтернативні потоки;
- X.5 постумови,

де X – число від 1 до кількості прецедентів.

Такий опис потрібен для аналізу діаграми і майбутнього програмування системи.

*Потік подій для прецеденту «Заява»*

*1.1. Передумови*

Повинна виконуватись тільки після підкріплення до прецеденту «Наказ», при чому видалення заяви буде після цього неможливим.

*1.2. Головний потік*

Прецедент починає виконуватися, коли фахівець підключається до системи і вводить своє ім'я і пароль. Система перевіряє правильність пароля і виводить можливі варіанти дій: додати, змінити, переглянути чи вийти. Якщо вибрана операція додати, S-1: виконується потік додати нову заяву.

Якщо вибрана операція змінити, S-2: виконується потік змінити даних в заяві (про заяву).

Якщо вибрана операція переглянути, S-3: виконується потік переглянути дані про заяву.

Якщо вибрана операція вийти, прецедент завершується.

*1.3. Підпотоки*

S-1: додати нову заяву.

Система відображає вікно діалогу, яке містить поля вводу даних для нової заяви. Фахівець заповнює поля: кому належить заява, дата написання заяви, на чиє ім'я, призначення, на який термін, кількість ставки і т.і. Система запам'ятовує введені дані. Далі заява виводиться на друк. Потім прецедент розпочинається спочатку.

S-2: змінити дані заяви

Система відображає вікно діалогу, яке містить список заяв і поле для вводу номера (ідентифікатора) заяви. Фахівець вибирає необхідну заяву зі списку чи вводить її номер (ідентифікатор) у відповідне поле. Система відображає інформацію про дану заяву. Фахівець вносить потрібні зміни. Система запам'ятовує введені дані. Потім прецедент розпочинається спочатку.

S-3: переглянути дані про заяву

Система відображає вікно діалогу, що містить список заяв і поле для вводу номера (ідентифікатора) заяви. Менеджер вибирає необхідну заяву зі списку чи вводить її номер (ідентифікатор) у відповідне поле. Система відображає інформацію про заяву (саму заяву). Коли фахівець перегляне інформацію, прецедент розпочнеться спочатку.

Видалення заяви можливе до того моменту поки вона не підкріплена до наказу.

*1.4. Альтернативні потоки*

1: введено неправильне ім'я чи пароль. Користувач повинен повторити введення чи завершити прецедент.

2: заповнені не всі поля. Фахівець повинен заповнити пусті поля чи завершити прецедент.

3: введено неправильний номер (ідентифікатор) заяви. Фахівець повинен повторити введення чи завершити прецедент.

За аналогічним сценарієм мають бути описані потоки подій для інших прецедентів.

Після опису моделі прецедентів переходимо до побудови та опису наступних моделей.

**Висновки.** Обсяги і складність сучасних інформаційних систем призвели до того, що UML- проектування виявляється обов'язковою складовою процесу розробки й передує етапу реалізації. Отримані в такий спосіб діаграми є основою проектованого програмного забезпечення й подаються у вигляді формальних інформаційних моделей. Розглянутий фрагмент прикладу показує лише деякі прийоми проектування програмного забезпечення за допомогою методів UML.

**БІБЛІОГРАФІЯ**

1. Винчугова А.А. Методы и средства концептуального проектирования информационных систем: сравнительный анализ стуктурного и объектно-ориентированого подходов / А.А.Винчугова // Прикладная информатика №1(49). - 2014. - С.56-66

2. Буч Г. UML. Классика CS. 2-е изд. / Пер. с англ.; Под общей ред. проф. С. Орлова / Г. Буч, А. Якобсон, Дж. Рамбо. – СПб.: Питер, 2006. – 736 с.

3.Гагарина Л. Г.Технология разработки программного обеспечения / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул. - М.: Форум, Инфра-М, 2008. – 402 с.

4.Erwin [Електронний ресурс] - Режим доступу: http://www.erwin. ru
5.MS Visio [Електронний ресурс] - Режим доступу: http://office.microsoft.com/en-us/FX010857981033.aspx
6.Dia [Електронний ресурс] - Режим доступу: http://live.gnome.org/Dia
7.Enterprise Architect [Електронний ресурс] - Режим доступу: http://www.sparxsystems.com /products/ea/index.html
8.Umbrello [Електронний ресурс] - Режим доступу: http://uml.sourceforge.net/
9.ArgoUML [Електронний ресурс] - Режим доступу: http://argouml.tigris.org/
10.UModel [Електронний ресурс] - Режим доступу: http://www.altova.com/products /umodel/umltool.html
11.Rational Rose [Електронний ресурс] - Режим доступу: http://www-01.ibm.com/software /rational/
12.Visual Paradigm [Електронний ресурс] - Режим доступу: http://visual-paradigm.com/

**ВІДОМОСТІ ПРО АВТОРА**

**Гриценко Валерій Григорович** – кандидат педагогічних наук, доцент, докторант Інституту ІТЗН НАПН України.

*Коло наукових інтересів:* ІКТ в освіті.

# ПРО РЕАЛІЗАЦІЮ ІНТЕГРАТИВНОГО НАВЧАЛЬНОГО КУРСУ У ПРОЦЕСІ ПІДГОТОВКИ МАЙБУТНІХ УЧИТЕЛІВ МАТЕМАТИКИ

***Вікторія НІЧИШИНА***

*У статті розкриваються мета, завдання та зміст програми навчального курсу «Інтеграція професійних знань майбутніх учителів математики». Розглядаються ефективні для реалізації змісту програми інтегративного навчального курсу організаційні форми.*

*This article reveals the goal, objectives and content of the program of the training course "Integration of professional knowledge of future teachers of Mathematics". It examines organizational forms, that are effective for the implementation of program content of integrative training courses.*

**Постановка проблеми.** Серед нагальних завдань, які ставляться державою перед системою освіти, актуальним на сьогоднішній день є завдання виховання громадян країни, суспільство якої матиме багату духовну культуру. Одним із можливих шляхів духовного насичення системи освіти є її гуманітаризація. Гуманітаризація змісту освіти передбачає радикальний перегляд змістового наповнення навчальних предметів. Це, в свою чергу, веде до перегляду і нової оцінки місця та ролі природничо-математичних дисциплін у навчальних закладах, адже математика і природничо-наукові предмети в процесі підготовки фахівців мають значний гуманітарний потенціал, який традиційно мало використовується. Одним із способів гуманітаризації математики і природничо-наукових дисциплін є їх інтеграція – органічне поєднання частини навчального матеріалу, при якому ускладнюються істотні зв'язки між його частинами.

Аналіз змісту та основних закономірностей традиційної методики професійної підготовки майбутніх учителів математики вказує на ряд існуючих недоречностей, які виникають через неузгодження, а в ряді випадків і через природню неможливість узгодження складових компонентів дисциплін природничо-наукового циклу підготовки майбутніх учителів математики (математичний аналіз, вища алгебра, аналітична геометрія, тощо). Це виражається:

– у недоречному подрібненні дидактичних одиниць знань в межах окремо