

УДК 004.62

А.М. Мельник, Р.М. Пасічник, Р.П. Шевчук

Тернопільський національний економічний університет, Тернопіль

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ТЕСТОВИХ ЗАВДАНЬ З КЕРОВАНОЮ СКЛАДНІСТЮ

В статті розвинуто методи та засоби автоматичної генерації тестових завдань різних типів, які базуються на основі системи семантичних класів, а також показано можливість апіорної оцінки складності цих завдань. Запропоновані методи враховують семантичні та формальні характеристики навчальних матеріалів, що дозволило підвищити якість сформованих завдань у порівнянні з відомими рішеннями.

Ключові слова: інформаційні технології, семантичний контроль, ефективність навчального процесу, автоматична генерація тестових завдань, управління складністю.

Вступ

Інтеграційні процеси, притаманні сучасному етапу розвитку суспільства, створення новітніх технологій, ускладнення технічних систем та інше потребує підвищення якості підготовки фахівців для народного господарства України [1]. Одним із засобів досягнення високого рівня цієї підготовки є розробка та впровадження інформаційної технології автоматичної генерації тестових завдань, яка включає методи та засоби генерування завдань закритого та відкритого типів, а також управління їх складністю. Це дозволить економити часові ресурси при формуванні бази диференційованих тестових завдань, а також сприятиме підвищенню підготовки навчальних матеріалів.

Аналіз досліджень та публікацій. Різноманітним аспектам тестування, створення і застосування навчальних і тестувальних систем на базі сучасних інформаційних технологій, питанням розробки баз даних і баз знань програмних систем присвячені численні праці українських та зарубіжних науковців: Аванесова В.С., Башмакова І.А. Клайна П., Гагаріна О.О., Титенка С.В., Пасічника В.В., Тонкононого В.М., Brusilovsky P., Schwarz, Weber. Більшість з них здійснювали дослідження в сфері проведення тестувань, наповнення бази тестових завдань за допомогою засобів підтримки ручного створення тестових завдань, безпеки процесу тестування і відтворення результатів. Особливо необхідно відзначити дослідження в напрямку автоматизації створення тестових завдань, оскільки цей напрямок, зважаючи є одним з найактуальніших напрямків досліджень [2]. Серед

відомих засобів автоматичної генерації тестових завдань необхідно відзначити методи параметризованих тестів, використання семантичних мереж а також використання понятійно-тезисної моделі. Основними недоліками даних підходів є створення завдань з низькою педагогічною цінністю, відсутність аналізу складності завдань, а також значні затрати на підготовку вхідної інформації. Тому виникає задача розробки інформаційної технології, яка направлена на розв'язання цих проблем [2].

Формалізація навчальних матеріалів та структури тестових завдань

Формалізація навчальних матеріалів та автоматична генерація тестових завдань відбувається на основі системи семантичних класів, яка запропонована в роботах [3, 4]. Інформативні матеріали (знання) представляються у вигляді як текстових тверджень, так і різноманітних схем, алгоритмів.

Текст предметної області поділяється на сукупність тверджень. З метою підвищення варіативності генерованих тестових завдань кожне твердження розбиваємо на компоненти, які описують процеси, поняття або їх характеристики. Компоненти поєднуються в речення сполучниками або сполучними словами у деревовидну структуру - відношення (1):

$$Df = \langle IdDf, CS, CNJ, TCNJ, Pr_IdDf, IdSc, IdFc \rangle, (1)$$

де Df – текстове твердження, яке формалізує певну сутність конкретного семантичного класу; $IdDf$ – ідентифікатор компоненти твердження; CS – зміст компоненти; CNJ – представлення зв'язку між компонентами твердження; $TCNJ$ – тип зв'язку між

компонентами; Pr_IdDf – посилання на батьківський елемент дерева (ідентифікатор відповідної компоненти); IdSc – ідентифікатор семантичного класу; IdFc – ідентифікатор формального класу.

Схема алгоритму дозволяє описати послідовність обов'язкових та необов'язкових операцій в ході реалізації мети - відношення (2) :

$$Sm = \left\langle IdSm, Cp, CS, Pr_IdSm, \right. \\ \left. SN, IdSc, IdFc, Lg_Int \right\rangle, \quad (2)$$

де Sm – схема методу; IdSm – ідентифікатор компоненти схеми методу; Cp – ідентифікатор обов'язковості елемента; CS – компонента схеми; Pr_IdSm – посилання на батьківський елемент; SN – назва методу; IdSc – ідентифікатор семантичного класу; IdFc – ідентифікатор формального класу; Lg_Int – лінгвістична інтерпретація операцій.

Метод автоматичної генерації тестового завдання «множинного» типу

Виходячи із формалізації інформативних матеріалів, яка описана відношеннями (1), (2) пропонується метод автоматичної генерації тестових завдань множинного типу. Основи даного підходу описані в роботах [2, 3]. Генерування завдань відбувається шляхом маніпулювання відповідними компонентами структур формалізованих семантичних класів, які семантично та формально узгоджені. Цей процес описується наступними кроками:

1. Генеруємо BT_{t1} першу частину основи тестового завдання BT_t =< BT_{t1}, BT_{t2} > :

$$BT_{t1_C} = \pi_{IdE, CS, CNJ} \\ (\sigma_{IdCh=IdCh_C \wedge IdSc=IdSc_C \wedge IdFc=IdFc_C \wedge IdE \notin SIdE \wedge Pr_IdE=0} \\ (\tau_{RANDLIMIT(1)}(Df))), \quad (3)$$

де Rand – функція випадкового вибору елемента множини; SIdE =< IdE > – множина ідентифікаторів вибраних основ тестових завдань; SIdE = {∅} перед початком процедури генерації тестових завдань, на кожному наступному кроці:

$$SIdE = SIdE \cup \pi_{IdE}(BT_{t1_C}). \quad (4)$$

2. Генеруємо другу частину основи тестового завдання Br із вірною альтернативою:

$$Br = \pi_{IdE, CS, CNJ, Pr_IdE} \gamma_{COUNT(IdE) ASlc} \\ (\sigma_{IdCh=IdCh_C \wedge IdSc=IdSc_C \wedge IdFc=IdFc_C \wedge Pr_IdE= \\ =\pi_{IdE}(BT_{t1_C})} \\ (\tau_{RANDLIMIT(1)}(Df))) \cup \pi_{IdE, CS, CNJ, Pr_IdE} \gamma_{COUNT(IdE) ASlc} \\ (\tau_{RANDLIMIT(1)}(Df ASDF1)) \\ \triangleright \triangleleft \\ Df1(Pr_IdE)=Df2(IdE) \\ \sigma_{Pr_IdE=\pi_{IdE}(BT_{t1_C})}(Df ASDF2), \quad (5)$$

де lc – рівень елемента Br.

3. Знаходимо висоту побудованої гілки дерева вибраного формалізованого твердження:

$$Brh = \gamma_{MAX(lc)}(Br). \quad (6)$$

4. Визначаємо висоту другої частини основи тесту BT_{t2h} :

$$BT_{t2h} = Rand(Brh - 1) \quad (7)$$

та ідентифікатор IdBk кінцевого вузла компоненти BT_{t2} _C :

$$IdBk = \pi_{IdE}(\sigma_{lc=BT_{t2h}}(Br)). \quad (8)$$

5. Генеруємо BT_{t2} другу частину основи тестового завдання:

$$BT_{t2_C} = \pi_{IdE, CS, CNJ}(\sigma_{IdE \leq IdBk}(Br)), \quad (9)$$

тоді основа тестового завдання:

$$BT_t_C = BT_{t1_C} \cup BT_{t2_C}. \quad (10)$$

6. Генеруємо множину правильних альтернатив:

6.1 Будуємо множину DBT_{t2} дочірніх елементів кінцевого вузла компоненти BT_{t2} _C :

$$DBT_{t2} = \pi_{IdE, CS, CNJ}(\sigma_{Pr_IdE=Id_Bk}(Df)). \quad (11)$$

6.2 Знаходимо кількість Dc дочірніх елементів кінцевого вузла компоненти BT_{t2} _C :

$$Dc = \gamma_{COUNT(IdE)}(DBT_{t2}). \quad (12)$$

6.3 Визначаємо кількість правильних альтернатив CtA тестового завдання:

$$CtA = Rand(Dc). \quad (13)$$

6.4 Будуємо початкові вузли правильних альтернатив StA₀ :

$$StA_0 = \pi_{IdE} \tau_{RANDLIMIT(Dc)}(DBT_{t2}); \quad (14)$$

$$StA = \pi_{IdE, CS, CNJ}(\sigma_{IdE=StA_0}(Df)) \cup (\pi_{IdE, CS, CNJ} \\ (\sigma_{IdE=\pi_{IdE}(StA_0) \vee Pr_IdE=\pi_{IdE}(StA_0)}(Df))). \quad (15)$$

7. Генерація хибних альтернатив.

7.1 Кількість хибних альтернатив:

$$CfA = CSa - CtA; \quad (16)$$

$$SfA = \pi_{IdE, CS, CNJ, Pr_IdE}$$

$$(\sigma_{IdCh=IdCh_C \wedge IdSc=IdSc_C \wedge IdFc=IdFc_C \wedge CNJ_C \wedge Pr_IdE \neq \pi_{IdE}(BT_{t1_C})} \\ (\tau_{RANDLIMIT(1)}(Df))) \cup \pi_{IdE, CS, CNJ, Pr_IdE} \\ (\tau_{RANDLIMIT(1)}(Df ASDF1)) \\ \triangleright \triangleleft \\ Df1(Pr_IdE)=Df2(IdE) \\ \sigma_{Pr_IdE=\pi_{IdE}(BT_{t1_C})}(Df ASDF2)). \quad (17)$$

7.2 Випадкове перемішування згенерованих альтернатив в тестовому завданні:

$$CSa = Rand(CtA, CfA). \quad (18)$$

8. Перевірка правильності вибраних альтернатив в процесі проведення тестового контролю:

8.1 Перевіряємо які альтернативи були вибрані, як відповідь:

$$CSa_C = \pi_{IdN}(CSa), \quad (19)$$

де IdN – ідентифікатор відзначеної альтернативи тестового завдання;

8.2 Проводимо перевірку кожної вибраної альтернативи на правильність, використовуючи наступне правило:

$$\text{якщо } \forall CSa_C \in CtA, \quad (20)$$

то відповідь на тестове завдання правильна;

$$\text{якщо } \exists CSa_C \in CtA \wedge \exists CSa_C \in CfA, \quad (21)$$

то відповідь на тестове завдання правильна частково;

$$\text{якщо } \forall CSa_C \notin CtA \wedge \forall CSa_C \in CfA, \quad (22)$$

то відповідь на тестове завдання не правильна.

9. Рекомендації щодо вивчення навчальних тем, по яких були допущені помилки в процесі тестування:

9.1 Із множини усіх вибраних хибних альтернатив CSaf_C знаходимо розділи Chf, по яких були допущені помилки:

$$CSaf_C = \forall CSa_C \in CfA; \quad (23)$$

$$Chf_C = \pi_{IdCh, Ch(NmCh)} \gamma_{COUNT(IdCh)ASEh} \quad (24)$$

$$(\sigma_{IdE=\pi_{IdE}(CSaf_C) \wedge IdCh=Ch(Id)}(Df, Ch),$$

де Ch – структура навчальних розділів, яка представлена як

$$Ch = \langle IdCh, NmCh, Pr_IdCh \rangle, \quad (25)$$

де IdCh – ідентифікатор розділу навчальної теми; NmCh – назва розділу; Pr_IdCh – ідентифікатор батьківського розділу.

9.2 Знаходимо розділи та відсортовуємо їх по найбільшій кількості помилок:

$$Chf = \tau_{Eh}(Chf_C). \quad (26)$$

Формалізація задач для їх програмної обробки

Поряд із задачами перевірки засвоєння теоретичних знань виникає задача оцінки вмій та навиків їх практичного використання. Метод автоматичної генерації задач передбачає варіативність умов задачі, яка забезпечується генерацією текстового опису задачі, вхідних змінних та їх значень. Також необхідно уникнути шаблонності розв'язків за рахунок виконання завдань на програмному інтерпретаторі як студентом так і системою.

Виходячи із поставлених завдань структура задач «відкритого» тесту формалізована за допомогою наступних відношень:

$$Task = \langle Condition, Inp_Value, Res_Value \rangle, \quad (27)$$

де Condition – умова задачі; Inp_Value – вхідні дані; Res_Value – результуючі змінні;

$$Condition = \langle Ft, Tf, Tfil \rangle, \quad (28)$$

де Ft – множина представлень умови задачі; Tf – множина допустимих типів значень змінних; Tfil – множина текстових представлень для уточнення умови задачі;

$$Ft = \langle IdFt, NmF, Lg_Int, IdCh, Tcv, IdTf, IdVtv \rangle, \quad (29)$$

де IdFt – ідентифікатор варіанту представлення; NmF – функція, яка реалізована в середовищі програмування; Lg_Int – лінгвістична інтерпретація NmF; Tcv – кількість вхідних змінних; IdTf – ідентифікатор типу змінних; IdVtv – ідентифікатор виду змінних.

$$Tf = \langle IdTf, Type \rangle, \quad (30)$$

де Type – тип значень вхідних та результуючих змінних, наприклад string, integer, тощо;

$$Vtv = \langle IdVtv, TypeVtv \rangle, \quad (31)$$

де TypeVtv – вид вхідних та результуючих змінних, наприклад масив, число, тощо;

$$Tfil = \langle IdTfil, Nmfil, Lg_Int, Tcv, IdTf, IdVtv \rangle, \quad (32)$$

де Nmfil – функція, яка реалізована в конкретному середовищі програмування;

$$Op = \langle IdOp, Oper, Lg_Oper \rangle, \quad (33)$$

де Oper – назва операції; Lg_Oper – лінгвістична інтерпретація операцій Oper;

$$Inp_Value = \langle IdV, Value, IdTf, IdVtv \rangle; \quad (34)$$

$$Res_Value = \langle Id, Value, IdTf \rangle; \quad (35)$$

$$Task = \langle Ft, Tf, Tfil, Inp_Value, Res_Value \rangle. \quad (36)$$

Метод автоматичної генерації задач з програмованим оператором

Метод автоматичної генерації множини задач з програмованим оператором передбачає генерацію задач на природній мові для студента (38) – (41), та відповідне завдання для інтерпретатора (42) – (44). При порівнянні результатів виконання програм згенерованих системою та створених студентом встановлюється адекватність розв'язку.

1. Генерація умови задачі Task :

$$Task = \langle TaskS, TaskI \rangle, \quad (37)$$

де TaskS – умова задачі для студента, TaskI – умова задачі для інтерпретатора.

2. Генерація задачі для студента TaskS :

2.1 формуємо умову задачі для студента на природній мові:

$$TaskS_C = \pi_{IdFt, IdCh, Lg_Int, Tcv, IdTf, IdVtv} (\sigma_{IdCh=IdCh_C}(\tau_{RANDLIMIT(1)}(Ft))). \quad (38)$$

2.2 Доповнюємо умову задачі вхідними змінними TaskIv :

$$TaskIv = \pi_{IdV, Value} (\sigma_{Inp_Value(IdTf)=TaskS_C(IdTf) \wedge Inp_Value(IdVtv)=TaskS_C(IdVtv)} (\tau_{RANDLIMIT(1)}(TaskS_C(Tcv)(TaskS_C, Inp_Value))). \quad (39)$$

2.3 Уточнюємо умову задачі TaskSp :

$$TaskSp = \pi_{IdTfil, Nmfil, Lg_Int, Tcv, IdTf, IdVtv} (\sigma_{TaskSp(IdTf)=TaskS_C(IdTf) \wedge TaskSp(IdVtv)=TaskS_C(IdVtv)} (\tau_{RANDLIMIT(1)}(Tfil, TaskS_C))). \quad (40)$$

2.4 Виводимо завдання для студента:

$$\begin{aligned} \text{TaskS} &= \text{CONCAT}(\text{TaskS_C}(\text{Lg_Int}), \\ &\text{TaskIv}(\text{Value}), \text{TaskSp}(\text{Lg_Int})). \end{aligned} \quad (41)$$

3. Генеруємо завдання для інтерпретатора TaskI :

$$\begin{aligned} \text{TaskI_C} &= \pi_{\text{IdFt}, \text{IdCh}, \text{NmF}, \text{Tcv}, \text{IdTf}, \text{IdVtv}} \\ &(\sigma_{\text{Ft}(\text{IdFt})=\text{TaskS_C}(\text{IdFt})}(\text{Ft}, \text{TaskS_C}))). \end{aligned} \quad (42)$$

3.1 Доповнюємо умову задачі контрольними вхідними змінними TaskCIv :

$$\begin{aligned} \text{TaskCIv} &= \pi_{\text{IdV}, \text{Value}} \\ &(\sigma_{\text{Inp_Value}(\text{IdTf})=\text{TaskS_C}(\text{IdTf})} \\ &\wedge \text{Inp_Value}(\text{IdVtv})=\text{TaskS_C}(\text{IdVtv}) \\ &\wedge \text{TaskIv}(\text{IdV}) \neq \text{Inp_Value}(\text{IdV})) \end{aligned} \quad (43)$$

$$\begin{aligned} &(\tau_{\text{RANDLIMIT}(\text{TaskS_C}(\text{Tcv}))} \\ &(\text{TaskS_C}, \text{Inp_Value}, \text{TaskIv}))). \end{aligned}$$

3.2 Формуємо завдання для інтерпретатора:

$$\begin{aligned} \text{TaskI} &= \text{CONCAT}(\text{TaskI_C}(\text{Lg_Int}), \\ &\text{TaskCIv}(\text{Value}), \text{TaskSp}(\text{Nmfil})). \end{aligned} \quad (44)$$

3.3 Фіксуємо результат його виконання:

$$\text{ResT} = \text{Interp}(\text{TaskI}). \quad (45)$$

4. Фіксуємо текст програми ProgS для інтерпретатора, сформовану студентом.

5. Фіксуємо результат виконання програми студента на контрольних даних.

$$\text{ResS} = \text{Interp}(\{\text{ProgS}, \text{TaskCIv}(\text{Value})\}). \quad (46)$$

6. Якщо $\text{ResS} = \text{ResT}$ – то задача розв’язана вірно.

Адаптивно-структурний метод оцінки складності тестових завдань

Питання автоматизованого тестового контролю супроводжується проблемами суб’єктивності оцінки складності самих завдань. Розподіл завдань по рівнях складності викладачем вносить певний суб’єктивізм в процес оцінки знань, оскільки не кожне легке завдання для викладача є легким і для студента. Відомі підходи оцінки складності тестових завдань ґрунтуються або на статистичних методах оцінки або на методах структурного аналізу. Статистичні методи вимагають великої вибірки, а методи на основі структурного аналізу – вимагають суб’єктивних експертних оцінок. Тому необхідним є створення адаптивно-структурного методу оцінки складності, що поєднував би переваги структурного аналізу та статистичних методів.

З класичної теорії тестів та сучасної теорії тестування IRT відомо, що складність тестового завдання залежить від правдоподібності неправильних відповідей, тобто чим краще підібрані дистрактори (неправильні відповіді) тим краще і складніше завдання. Встановлено, що вища частка вибору неправильних відповідей в процесі тестування, свідчить про вищу якість формування тесту.

Нехай тестові твердження представляються наступним чином:

$$O_i \rightarrow \{A_{i1}, \dots, A_{iN}\}, \quad (47)$$

де O_i – основна частина тестового твердження; A_{ij} – альтернативні специфікації тестового твердження; N – кількість альтернатив.

Оцінку складності альтернатив здійснимо на основі показника композиційної близькості D_{ij} .

Експериментально встановлено, що на складність ідентифікації альтернатив впливає близькість за тематикою та близькість за формою подання. Оцінку складності альтернатив за цим принципом здійснимо на основі показника композиційної близькості D_{ij} , співвідношення (48) та автентичності (49):

$$D_{ij} = \begin{cases} 1 - \text{композиційна віддаленість}; \\ 1 + \beta_1 - \text{відносна близькість}; \\ 1 + 2\beta_1 - \text{композиційна близькість}, \end{cases} \quad (48)$$

де β_1 – градація впливу близькості;

$$A_{ij} = \begin{cases} \alpha_0 - \text{заздалегідь невірна} \\ \text{альтернатива}; \\ \alpha_0 + \beta_2 - \text{невірна альтернатива,} \\ \text{що близька до вірної}; \\ \alpha_0 + 2\beta_2 - \text{вірна альтернатива}, \end{cases} \quad (49)$$

де α_0 – оцінка сили впливу кількості заздалегідь невірних альтернатив на її складність; β_2 – градація достовірності; N – кількість альтернатив.

Значення функцій D_{ij} і A_{ij} формуються за допомогою рівномірних шкал, масштаби яких β_1 та β_2 необхідно ідентифікувати (параметрично).

Складність ідентифікації альтернатив визначається взаємовпливом їх композиційної близькості та автентичності, а складність ідентифікації твердження зростає із збільшенням кількості альтернатив. В результаті отримуємо адитивно-мультиплікативну модель складності твердження (50). Оцінка складності твердження коректується за рахунок врахування INV - негативного представлення основної частини твердження (50).

$$CS_i^*(\vec{\beta}) = \sum_j D_{ij}(\beta_1) \cdot A_{ij}(\beta_2) + \text{INV}_i(\beta_2). \quad (50)$$

Експериментальні дослідження

Інформаційна технологія автоматичної генерації тестових завдань з керованою складністю лягла в основу веб-орієнтованої системи SAGT, яка використовується на факультеті комп’ютерних інформаційних технологій Тернопільського національного економічного університету. На рис. 1 представлено приклад згенерованого завдання множинного типу з дисципліни «Основи C++», а на рис. 2, приклад згенерованого завдання з програмованим оператором для перевірки знань з «PHP».

Вказівник -дозволяє отримати значення які

- ☐ зібрані певним чином
- ☐ будь-де в пам'яті
- ☐ дозволяють ідентифікувати певну структуру

Рис. 1. Автоматично згенероване тестове завдання з дисципліни «Основи C++»

Задано наступне завдання, необхідно написати програму для його вирішення

1. Знайти суму значень елементів масиву \$mass = [0 -7 -12 -15 -16 -15 -12 -7 0 9 20] серед додатних, а результат записати у змінну **resultat**

Викно для написання програми:

```
<? php
// вхідний масив
$mass = array (0, -7, -12, -15, -16, -15, -12, -7, 0, 9, 20);
// функція пошуку додатних елементів
function dodat($var) {
    return ($var % 2 > 0);
}
// відфільтрований масив
$array=array_filter($mass, "dodat");
// пошук шуканого розв'язку
$result=array_sum($array);
echo $result;
?>
```

Перевірити результат

Рис. 2. Згенерована задача з «PHP»

Висновки

В процесі проведених досліджень отримано наступні наукові та практичні результати:

– запропоновано інформаційну технологію автоматичної генерації тестових завдань, яка включає методи та засоби генерування завдань закритого та відкритого типів, а також адаптивно-структурний метод апіорної оцінки складності цих завдань, яка на відміну від відомих рішень дозволяє суттєво економити часові ресурси в процесі побудови тестових завдань, а також управляти їх складністю.

Список літератури

1. Бадьорина Л.М. Інформаційна технологія при-
родного контролю знань: автореф. дис. ... канд.
техн. наук: 05.13.06 / Л.М. Бадьорина. – К.: Київський
національний університет будівництва і архітектури,
2009. – 17 с.

2. Мельник А.М. Автоматична генерація тестових
завдань різних типів / А.М. Мельник // Вісник Хмельниць-
кого національного університету. – 2010. – № 4. – С. 124-
129.

3. Мельник А.М. Метод генерації тестових завдань
на основі системи семантичних класів / А.М. Мельник,
Р.М. Пасічник // Вісник ТДТУ. – 2010. – Т. 15, № 1. –
С. 187-193.

4. Melnyk A. System of semantic classes for test's
generation. Modern Problems of Radio Engineering,
Telecommunications and Computer Science / A. Melnyk,
R. Pasichnyk // Proceedings of the International Conference
TSCET'2010. Lviv-Svavsk, Ukraine February 23-27, 2010. –
P. 206-207.

Надійшла до редколегії 7.04.2011

Рецензент: д-р фіз.-мат. наук, проф. Д.І. Боднар, Терно-
пільський національний економічний університет, Терно-
піль.

ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ТЕСТОВЫХ ЗАДАНИЙ С УПРАВЛЯЮЩЕЙ СЛОЖНОСТЬЮ

А.Н. Мельник, Р.М. Пасичник, Р.П. Шевчук

В статье развиты методы и средства автоматической генерации тестовых заданий различных типов, которые базируются на основе системы семантических классов, а также показана возможность априорной оценки сложности этих задач. Предложенные методы учитывают семантические и формальные характеристики учебных материалов, что позволило повысить качество сформированных задач по сравнению с известными решениями.

Ключевые слова: информационные технологии, семантический контроль, эффективность учебного процесса, автоматическая генерация тестовых заданий, управление сложностью.

INFORMATION TECHNOLOGY AUTOMATIC GENERATION OF TESTS WITH CONTROLLED COMPLEXITY

A.M. Melnyk, R.M. Pasichnyk, R.P. Shevchuk

In this paper the problem of automated test's generation are investigated, information technology for generating test tasks of various types of differentiated complexity are proposed.

Keywords: Information technology, semantic control, efficiency of the educational process, the automatic generation of tests, management of complexity.