решения конкретных задач профилактики, диагностики и лечения функциональных нарушений и анатомо-физиологических отклонений жизнедеятельности человеческого организма, а также ценностно-мотивационных установок и качеств, необходимых для осуществления клинической деятельности, проведения медицинских научных исследований, овладения новыми видами медицинских методик, технологий и техники.

Результатом практико-ориентированной естественно-научной подготовки будущих специалистов в сфере клинической медицины является естественно-научная компетентность как неотъемлемая составляющая их профессиональной компетентности. Естественно-научную компетентность будущего врача-клинициста мы определяем, как динамичное личностное образование, которое интегрирует естественно-научные знания, умения, навыки и опыт их использования в диагностической, лечебной, профилактической и научно-исследовательской деятельности врача-клинициста, обуславливая его способность и готовность решать разнообразные проблемы медицины и здравоохранения, а также возможность овладения новыми видами медицинских методик, технологий и техники [3].

При определении сущности и структуры естественно-научной компетентности будущих медиков была произведена проекция требований профессиональных стандартов на систему естественно-научных знаний, умений и навыков, что позволит наполнить практико-ориентированном смыслом содержание естественно-научной подготовки и сформировать способность будущих врачей-клиницистов к выполнению конкретных действий по диагностике, лечению и профилактике заболеваний, проведению научных медицинских исследований, работе с медицинским оборудованием и техникой.

Указанное подтверждает обоснованность выбора компетентностного подхода в качестве ведущего методологического ориентира практико-ориентированной естественно-научной подготовки будущих врачей.

Литература

1. Гельман, В. Я. Преподавание математических дисциплин в медицинском вузе / В. Я. Гельман, Л. А. Ушверидзе, Ю. П. Сердюков // Образование и наука. – 2018. – Т. 20, № 2. – С. 88–107.

2. Россомахина, О. М. Роль дисциплин естественнонаучного цикла в практико-ориентированной подготовке будущих специалистов в сфере клинической медицины / О. М. Россомахина // Донецкие чтения 2021: образование, наука, инновации, культура и вызовы современности : Материалы VI Международной научной конференции, Донецк, 26–28 октября 2021 года. Том 6 Часть 2. – Донецк : Донецкий национальный университет, 2021. – С. 212–215.

3. Россомахина, О. М. Сущность практико-ориентированной естественно-научной подготовки будущих специалистов в сфере клинической медицины / О. М. Россомахина // Ученые записки Орловского государственного университета. – 2022. – № 4(97). – С. 308–311.

4. Технологии приобретения компетенций при подготовке врача (опыт Казанского федерального университета) / Е.В. Киясова [и др.] // Медицинское образование и профессиональное развитие. – 2017. – №4. – С. 57–64.

УДК 004.05

Е. А. Ружицкая
г. Гомель, ГГУ имени Ф. Скорины

ИСПОЛЬЗОВАНИЕ ПРИНЦИПОВ ПРОЕКТИРОВАНИЯ ПРИ ПОДГОТОВКЕ IT-СПЕЦИАЛИСТОВ

Программирование – это искусство создания программного обеспечения, которое должно быть не только функциональным, но и понятным, поддерживаемым и гибким.

Для достижения этих целей разработчики следуют ряду принципов, которые помогают им создавать качественный код. Одной из наиболее известных групп принципов является SOLID. Эти пять принципов помогают разработчикам создавать более чистый и организованный код, что особенно важно при подготовке IT-специалистов. Рассмотрим, как принципы проектирования могут быть использованы в образовательных программах для подготовки будущих IT-специалистов, а также примеры, как эти принципы могут быть внедрены в учебные программы.

SOLID – это акроним, введённый Майклом Фэзерсом для первых пяти принципов, названных Робертом Мартином в начале 2000-х, который обозначает пять основных принципов объектно-ориентированного проектирования [1]:

S – Single Responsibility Principle (Принцип единственной ответственности);

O – Open / Closed Principle (Принцип открытости / закрытости);

L – Liskov Substitution Principle (Принцип подстановки Лисков);

I – Interface Segregation Principle (Принцип разделения интерфейса);

D – Dependency Inversion Principle (Принцип инверсии зависимостей).

Каждый из этих принципов играет важную роль в создании качественного кода и может быть интегрирован в учебные программы для IT-специалистов.

Принцип единственной ответственности (SRP) утверждает, что каждый класс должен выполнять только одну задачу. Если класс выполняет более одной задачи, это может привести к сложностям в его тестировании, сопровождении и изменении. В образовательных программах важно обучать студентов этому принципу, чтобы они понимали, как разбивать сложные системы на более простые и управляемые компоненты. Применение SRP помогает избежать избыточности и улучшает читаемость кода. Например, студенты могут работать над проектами, где им нужно разделить функциональность на несколько классов, каждый из которых отвечает за свою задачу. Это помогает развивать навыки проектирования и улучшает понимание архитектуры программного обеспечения. Разделение функциональности на узкоспециализированные классы помогает разработчикам легче понимать и изменять код. Например, в приложении для управления библиотекой можно выделить отдельные классы для управления книгами, пользователями и выдачей книг. Это упрощает тестирование и модификацию каждого из классов.

Принцип открытости / закрытости (OCP) утверждает, что программные сущности (классы, модули и функции) должны быть открыты для расширения, но закрыты для модификации. Это позволяет разработчикам добавлять новую функциональность без изменения существующего кода, что снижает риск возникновения ошибок. Например, студенты могут изучать паттерны проектирования, такие как «Стратегия» или «Наблюдатель», которые позволяют им создавать расширяемые системы, не внося изменения в уже существующий код. Например, можно создать интерфейс для методов сортировки и реализовать его в нескольких классах, каждый из которых будет представлять свою стратегию сортировки.

Принцип подстановки Лисков (LSP) гласит, что объекты подкласса должны быть взаимозаменяемыми с объектами суперкласса без изменения желаемых свойств программы. Это означает, что если класс S является подклассом класса T, то объекты типа T должны быть заменяемы объектами типа S без изменения желаемых свойств программы. Студенты могут работать с иерархиями классов и разрабатывать системы, где подклассы корректно наследуют поведение суперклассов. Например, если у студентов есть класс Bird, который имеет метод fly(), и они создают подкласс Penguin, который не может летать, это нарушает LSP. Вместо этого студенты могут создать интерфейс Flyable, который будет реализован только теми классами, которые действительно могут летать, избегая таким образом нарушения принципа.

Принцип разделения интерфейса (ISP) утверждает, что клиенты не должны зависеть от интерфейсов, которые они не используют. Это означает, что интерфейсы должны быть специализированными, а не общими. Это помогает избежать создания громоздких интерфейсов и улучшает модульность кода. Студенты изучают, как создавать небольшие и специализированные интерфейсы, которые предоставляют только необходимый функционал. Это помогает избежать создания громоздких интерфейсов и улучшает модульность кода. Например, в проекте по разработке системы для управления устройствами лучше создать отдельные интерфейсы для принтеров, сканеров и факсов, чем один общий интерфейс для всех устройств.

Принцип инверсии зависимостей (DIP) гласит, что высокоуровневые модули не должны зависеть от низкоуровневых. Оба типа модулей должны зависеть от абстракций (интерфейсов). Это позволяет уменьшить связанность между компонентами системы. Студенты изучают использование инверсии зависимостей и контейнеров для управления зависимостями в приложениях. Это помогает им создавать более гибкие и тестируемые системы. Например, в проекте, где разрабатывается система уведомлений, студенты могут создать интерфейс INotificationService, который будет реализован различными классами, такими как EmailNotificationService и SMSNotificationService. Это позволяет высокоуровневым модулям работать с абстракциями, а не с конкретными реализациями.

Принципы SOLID являются основополагающими, однако есть еще и другие, не менее важные принципы проектирования программного обеспечения. К ним относятся:

DRY (Don’t repeat yourself – «Не повторяйся») – принцип, направленный на снижение повторения информации различного рода [2].

KISS (“Keep it simple, stupid” – «Делай проще…») – принцип утверждает, что большинство систем работают лучше всего, если они остаются простыми, а не усложняются [3].

YAGNI (“You aren’t gonna need it” – «Вам это не понадобится») – процесс и принцип проектирования, при котором в качестве основной цели и/или ценности декларируется отказ добавления функциональности, в которой нет непосредственной надобности [4].

Бри́тва О́ккама – принцип, гласящий: «Не следует привлекать новые сущности без крайней на то необходимости».

Разделение ответственностей – это упрощение единого процесса решения задачи путём разделения на взаимодействующие процессы по решению подзадач.

Знание принципов проектирования программного обеспечения помогает разработчикам создавать более качественный, поддерживаемый и гибкий код. Следование этим принципам не только улучшает архитектуру программного обеспечения, но и облегчает его тестирование и модификацию. Важно, чтобы каждый IT-специалист понимал и применял эти принципы в своей практике, что в конечном итоге приведет к созданию более надежных и эффективных программных решений.

Литература

1. SOLID (программирование). – Режим доступа: https://ru.wikipedia.org/wiki/SOLID_(программирование). – Дата доступа: 29.01.2025.

2. Don't_repeat_yoursel. – Режим доступа: https://ru.wikipedia.org/wiki/Don't_repeat_yourself. – Дата доступа: 29.01.2025.

3. KISS_(принцип). – Режим доступа: https://ru.wikipedia.org/wiki/KISS_(принцип). – Дата доступа: 29.01.2025.

4. YAGNI. – Режим доступа: https://ru.wikipedia.org/wiki/YAGNI. – Дата доступа: 29.01.2025.