МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Лекция №5

Методы разработки информационных систем

Методы и средства проектирования информационно-аналитических систем

(наименование дисциплины (модуля) в соответствии с учебным планом)

Уровень: специалитет

(бакалавриат, магистратура, специалитет)

Форма обучения: очная

(очная, очно-заочная, заочная)

Направление(-я) подготовки: 10.05.04 «Информационно-аналитические системы безопасности»

(код(-ы) и наименование(-я))

Институт: Институт кибербезопасности и цифровых технологий (ИКБ)

(полное и краткое наименование)

Кафедра: Информационно-аналитические системы кибербезопасности (КБ-2)

(полное и краткое наименование кафедры, реализующей дисциплину (модуль))

Используются в данной редакции с учебного года 2023/24

(учебный год цифрами)

Проверено и согласовано «____» _____20___г.

(подпись директора Института/Филиала с расшифровкой)

Москва 2024 г.

Жизненный цикл — совокупность последовательно меняющихся состояний организации, каждое из которых соответствует определенному комплексу управленческих характеристик и типовой модели поведения компании. В свою очередь жизненный цикл программного обеспечения (ПО) является периодом времени, представляющий собой непрерывный процесс построения и развития информационной системы, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается его полным изъятием из эксплуатации. Жизненный цикл информационной системы включает в себя такие стадии, как планирование, анализ, проектирование, реализацию, внедрение и эксплуатацию [1]. Проектирование и испытания информационных систем регламентируются стандартами, ведущим из которых является международный стандарт ISO/IEC 12207. Рекомендации данного стандарта являются общими для различных моделей. Под моделью подразумевается структура, определяющая последовательность выполнения процессов, направленных на решение задач, возникающих на протяжении жизненного цикла программного обеспечения. В настоящее время известно большое количество моделей жизненного цикла, но наиболее распространенными являются: каскадная, прототипная, спиральная и др. Рассмотрим их поподробнее.

Каскадная модель

Классической моделью жизненного цикла (ЖЦ) программного обеспечения (ПО) информационной системы является каскадная модель. Это была первая модель, которая формализовала структуру этапов разработки ПО, придавая особое значение исходным требованиям и проектированию, а также созданию документации на ранних этапах процесса разработки. Схема каскадной модели ЖЦ ПО представлена на рис.

Рис. Каскадная модель жизненного цикла ПО

Внешне ее жизненный цикл напоминает поток воды водопада, который последовательно проходит через фазы выработки системных требований, выработки требований к ПО, анализа, проектирования, кодирования, тестирования, интеграции и поддержки [2]. Таким образом, можно сделать вывод, что сущность данной модели заключается в том, чтобы процесс создания программного продукта был разбит на последовательные этапы, в результате завершения которых будут формироваться промежуточные продукты, не способные изменяться на последующих шагах. Таким образом, главным недостатком данной модели является низкая гибкость в управлении проектом. Другими словами, если появится необходимость внести в проект какие-либо изменения, то это повлечет существенные дополнительные затраты для организации. К достоинствам данной модели можно отнести простоту применения, способность планировать сроки завершения всех работ,

соответствующие ресурсы и набор проектной документации, которая формируется после завершения каждого этапа. Следовательно, можно прийти к выводу, что наиболее эффективно применение каскадной модели при разработке проектов, требования которых не изменяются в течении жизненного цикла; при разработке проекта, ориентированного на построение системы или продукта такого же типа; при разработке проекта, связанного с переносом уже существующего продукта или системы на новую платформу [3].

Старейшая парадигма, разработал Уинстон Ройс, 1970.

Разработка рассматривается как последовательность этапов, причем переход на следующий, этап происходит только после полного завершения работ на текущем этапе

Содержание основных этапов.

Разработка начинается на системном уровне и проходит через анализ, проектирование, кодирование, тестирование и сопровождение. При этом моделируются действия стандартного инженерного цикла.

Системный анализ задает роль каждого элемента в компьютерной системе, взаимодействие элементов друг с другом. Анализ начинается с определения требований ко всем системным элементам и назначения подмножества этих требований программному «элементу». Необходимость системного подхода явно проявляется, когда формируется интерфейс ПО с другими элементами (аппаратурой, людьми, базами данных). На этом же этапе начинается решение задачи планирования проекта. В ходе планирования проекта определяются объем проектных работ и их риск, необходимые трудозатраты, формируются рабочие задачи и план-график работ.

Анализ требований относится к программному элементу — программному обеспечению. Уточняются и детализируются его функции, характеристики и интерфейс. Все определения документируются в спецификации анализа. Здесь же завершается решение задачи планирования проекта.

Проектирование состоит в создании представлений:

- архитектуры ПО;
- модульной структуры ПО;
- алгоритмической структуры ПО;
- структуры данных; входного и выходного интерфейса (входных и выходных форм данных).

Исходные данные для проектирования содержатся в спецификации анализа, то есть в ходе проектирования выполняется трансляция требований во множество проектных представлений. При решении задач проектирования основное внимание уделяется качеству будущего программного продукта.

Кодирование состоит в переводе результатов проектирования в текст на языке программирования.

Тестирование — выполнение программы для выявления дефектов в функциях, логике и форме реализации программного продукта.

Сопровождение — это внесение изменений в эксплуатируемое ПО с целью:

- исправление ошибок;
- адаптация к изменениям внешней для ПО среды;
- усовершенствование ПО по требованиям заказчика.

Сопровождение ПО состоит в повторном применении каждого из предшествующих шагов (этапов) жизненного цикла к существующей программе, но не в разработке новой программы.

Достоинства классического жизненного цикла:

- дает план и временной график по всем этапам проекта;
- упорядочивает ход конструирования.

Недостатки классического жизненного цикла:

- реальные проекты часто требуют отклонения от стандартной последовательности шагов;
- цикл основан на точной формулировке исходных требований к ПО (реально в начале проекта требования заказчика определены лишь частично);
- результаты проекта доступны заказчику только в конце работы.

Когда использовать каскадную методологию?

- Только тогда, когда требования известны, понятны и зафиксированы.
- Противоречивых требований не имеется.
- Нет проблем с доступностью программистов нужной квалификации.
- В относительно небольших проектах.

V-Model

Унаследовала структуру «шаг за шагом» от каскадной модели. V-образная модель применима к системам, которым особенно важно бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности в транспортных средствах и так далее. Особенностью модели можно считать то, что она направлена на тщательную проверку и тестирование продукта, находящегося уже на первоначальных стадиях проектирования. Стадия тестирования проводится одновременно с соответствующей стадией разработки, например, во время кодирования пишутся модульные тесты.

Рис. V-образная модель проектирования

Проект выполняется по четкому техническому заданию, но в него включен значительный этап тестирования: удобства интерфейса, функционального, нагрузочного и в том числе интеграционного, которое должно подтверждать, что несколько компонентов от различных производителей вместе работают стабильно, невозможны сбои, отказы, злоупотребления недостатками системы.

Когда использовать V-модель?:

- Если требуется тщательное тестирование продукта, то V-модель оправдает заложенную в себя идею: validation and verification.
- Для малых и средних проектов, где требования четко определены и фиксированы.
- В условиях доступности инженеров необходимой квалификации, особенно тестировщиков.

Макетирование (прототипирование)

В качестве следующей модели предлагаю рассмотреть модель прототипирования. Прототипирование — это процесс построения рабочей модели системы, в которой пользователь и программист разрабатывают предварительный план или модель проекта. На рис. показан процесс прототипирования, который основывается на многократном повторении итераций.

Рис. Процесс прототипирования

Ее особенностью является то, что заказчик начинает знакомиться с системой на ранних этапах разработки. При этом разработчик демонстрирует пользователям готовый макет, а пользователи оценивают его функционирование. После этого выявляются проблемы, над устранением которых совместно работают пользователи и разработчики. Как показано на рис. , данный процесс будет продолжаться до тех пор, пока пользователи не будут удовлетворены степенью соответствия программного продукта. Таким образом, главным преимуществом создания прототипов является возможность улучшить взаимопонимание между всеми участниками процесса и свести к минимуму риск получения продукта, не соответствующего выдвигаемым требованиям. Но, как и в любой модели жизненного цикла, есть и свои недостатки. Минусом данной модели является то, что существует риск затягивания фазы создания прототипов, и как следствие это способно привести к удорожанию проекта за счет включения незапланированных итераций. Прототипирование лучше всего применять в проектах, в которых заранее отсутствуют четко сформулированные требования, или же когда выполняется новая, не имеющая аналогов разработка.

Основная цель макетирования (прототипирования) — снять неопределенности в требованиях заказчика.

Макетирование (прототипирование) — это процесс создания модели требуемого программного продукта.

Модель может принимать одну из трех форм:

- бумажный макет или макет на основе ПК
- работающий макет (выполняет часть требуемых функций);
- существующая программа (характеристики которой затем должны быть улучшены).

Макетирование начинается со сбора и уточнения требований к создаваемому ПО. Затем выполняется быстрое проектирование. В нем внимание сосредоточивается на тех характеристиках ПО, которые должны быть видимы пользователю. Быстрое проектирование приводит к построению макета.

Макет оценивается заказчиком и используется для уточнения требований к ПО. Итерации повторяются до тех пор, пока макет не выявит все требования заказчика.

Достоинство макетирования: обеспечивает определение полных требований к ПО.

Недостатки макетирования:

- заказчик может принять макет за продукт;
- разработчик может принять макет за продукт.

Когда заказчик видит работающую версию ПО, он перестает сознавать, что детали макета скреплены «жевательной резинкой и проволокой»; он забывает, что в погоне за работающим вариантом оставлены нерешенными вопросы качества и удобства сопровождения ПО.

С другой стороны, для быстрого получения работающего макета разработчик часто идет на определенные компромиссы. Могут использоваться не самые подходящие языки программирования или операционные системы. Для простой демонстрации возможностей может применяться неэффективный алгоритм. Спустя некоторое время разработчик забывает о причинах, по которым эти средства не подходят. В результате далеко не идеальный выбранный вариант интегрируется в систему.

Инкрементная модель

Инкрементная модель является классическим примером инкрементной стратегии конструирования). Она объединяет элементы последовательной водопадной модели с итерационной философией макетирования.

Каждая линейная последовательность здесь вырабатывает поставляемый инкремент ПО. Первый инкремент приводит к получению базового продукта, реализующего базовые требования (правда, многие вспомогательные требования остаются нереализованными).

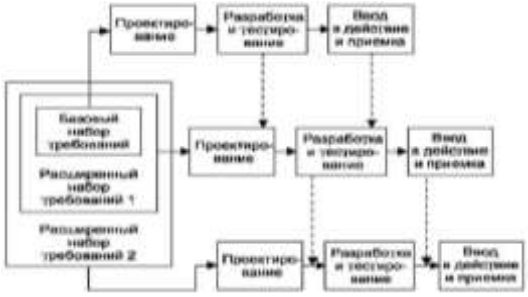

Рис. Инкрементная модель разработки

План следующего инкремента предусматривает модификацию базового продукта, обеспечивающую дополнительные характеристики и функциональность. По своей природе инкрементный процесс итеративен, но, в отличие от макетирования, инкрементная модель обеспечивает на каждом инкременте работающий продукт.

Когда использовать инкрементную модель?

- Когда основные требования к системе четко определены и понятны. В то же время некоторые детали могут дорабатываться с течением времени.
- Требуется ранний вывод продукта на рынок.
- Есть несколько рисковых фич или целей.

Быстрая разработка приложений

Модель быстрой разработки приложений (**Rapid Application Development**) — второй пример применения инкрементной стратегии конструирования.

RAD-модель обеспечивает экстремально короткий цикл разработки. RAD — высокоскоростная адаптация линейной последовательной модели, в которой быстрая разработка достигается за счет использования компонентно-ориентированного конструирования. Если требования полностью определены, а проектная область ограничена, RAD-процесс позволяет группе создать полностью функциональную систему за очень короткое время (60-90 дней). RAD-подход ориентирован на разработку информационных систем и выделяет следующие этапы:

бизнес-моделирование. Моделируется информационный поток между бизнес-функциями. Ищется ответ на следующие вопросы:

- Какая информация руководит бизнеспроцессом?
- Какая генерируется информация?
- Кто генерирует ее?
- Где информация применяется?
- Кто обрабатывает ее?

моделирование данных. Информационный поток, определенный на этапе бизнес-моделирования, отображается в набор объектов данных, которые требуются для поддержки бизнеса. Идентифицируются характеристики (свойства, атрибуты) каждого объекта, определяются отношения между объектами **моделирование обработки**. Определяются преобразования объектов данных, обеспечивающие реализацию бизнес-функций. Создаются описания обработки для добавления,

модификации, удаления или нахождения (исправления) объектов данных; **генерация приложения**. Предполагается использование методов, ориентированных *на языки программирования 4-го поколения*. Вместо создания ПО с *помощью языков программирования 3-го поколения,* RAD-процесс работает с повторно используемыми программными компонентами или создает повторно используемые компоненты. Для обеспечения конструирования используются утилиты автоматизации: **тестирование и объединение**. Поскольку применяются повторно используемые компоненты, многие программные элементы уже протестированы. Это уменьшает время тестирования (хотя все новые элементы должны быть протестированы).

Рис. Быстрая разработка

Применение RAD возможно в том случае, когда каждая главная функция может быть завершена за 3 месяца. Каждая главная функция адресуется отдельной группе разработчиков, а затем интегрируется в целую систему.

Применение RAD имеет и свои недостатки, и ограничения.

Для больших проектов в RAD требуются существенные людские ресурсы (необходимо создать достаточное количество групп).

RAD применима только для таких приложений, которые могут декомпозироваться на отдельные модули и в которых производительность не является критической величиной.

RAD не применима в условиях высоких технических рисков (то есть при использовании новой технологии).

Когда используется RAD-модель?

- Может использоваться только при наличии высококвалифицированных и узкоспециализированных архитекторов.
- Бюджет проекта большой, чтобы оплатить этих специалистов вместе со стоимостью готовых инструментов автоматизированной сборки.
- RAD-модель может быть выбрана при уверенном знании целевого бизнеса и необходимости срочного производства системы в течение 2-3 месяцев.

Спиральная модель жизненного цикла программного обеспечения.

Она является классическим примером эволюционной стратегии конструирования программного обеспечения (автор Барри Боэм, 1988). Ее отличительной особенностью является сочетание в себе возможности предыдущих моделей.

Спиральная модель предполагает 4 этапа для каждого витка:

- планирование;
- анализ рисков;
- конструирование;
- оценка результата и при удовлетворительном качестве переход к новому витку.

Так на этапах анализа и проектирования степень удовлетворения потребностей заказчика актуально проверять путем создания прототипов. Данное решение позволяет уточнить требования, цели и характеристики проекта, определить качество разработки, спланировать работы следующего витка спирали [4]. В последствии будет выбран такой вариант, который будет обоснованно удовлетворять действительным требованиям заказчика. К недостаткам данной модели следует отнести: дороговизну оценки рисков, так как оценка рисков после прохождения каждой спирали связана с большими затратами; усложненную структуру жизненного цикла; бесконечность спирали, это связано с тем, что каждая ответная реакция заказчика на созданную версию может порождать новый цикл, что затягивает завершение работы над проектом. Подробная схема данной модели представлена на рис.

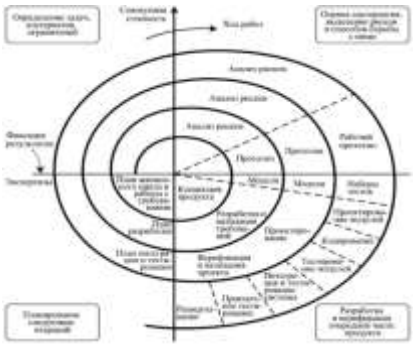

Рис. 3. Спиральная модель жизненного цикла ПО

Учитывая вышеуказанные преимущества и недостатки, можно прийти к выводу, что спиральную модель ЖЦ целесообразно применять при разработке долгосрочных проектов, при разработке проектов, использующих новые технологии, или проектов с ожидаемыми существенными изменениями.

модель не подойдет для малых проектов, она резонна для сложных и дорогих, например, таких, как разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование.

«Agile Model» (гибкая методология разработки)

В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике.

Extreme programming (XP) - это методология разработки программного обеспечения, направленная на повышение качества программного обеспечения и его оперативности в соответствии с меняющимися требованиями заказчика. Как разновидность гибкой разработки программного обеспечения, она выступает за частые "релизы" в коротких циклах разработки, что призвано повысить

производительность и ввести контрольные точки, на которых могут быть приняты новые требования клиентов.

В основе такого типа — непродолжительные ежедневные встречи — «Scrum» и регулярно повторяющиеся собрания (раз в неделю, раз в две недели или раз в месяц), которые называются «Sprint». На ежедневных совещаниях участники команды обсуждают:

- отчёт о проделанной работе с момента последнего Scrum'а;
- список задач, которые сотрудник должен выполнить до следующего собрания;
- затруднения, возникшие в ходе работы.

Методология подходит для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка. Соответственно, в процессе реализации требования изменяются. Стоит вспомнить класс творческих людей, которым свойственно генерировать, выдавать и опробовать новые идеи еженедельно или даже ежедневно. Гибкая разработка лучше всего подходит для этого психотипа руководителей.

Когда использовать Agile?

- Когда потребности пользователей постоянно меняются в динамическом бизнесе.
- Изменения на Agile реализуются за меньшую цену из-за частых инкрементов.
- В отличие от модели водопада, в гибкой модели для старта проекта достаточно лишь небольшого планирования.

Методика выбора на основе анализа критериев.

В основе данной методики лежит использование таблицы, где по заданным критериям даются качественные оценки трем моделям жизненного цикла — каскадной, прототипной, спиральной. Данная методика позволяет наглядно сравнить характеристики. Согласно данной методике, наиболее приемлемой моделью жизненного цикла будет являться та, в соответствующем столбце которой будет выбрано большее число оценок [5]. Сравнение характеристик, выбранных моделей жизненного цикла программного обеспечения, представлена в таблице 1.

Таблица 1 Иллюстрация методики выбора модели жизненного цикла на основе анализа критериев

Характеристика проекта	Модель		
	Каскадная	Прототипная	Спиральная
Новизна разработки и обеспеченность ресурсами	высокая	средняя	средняя

Длительность разработки	до года	до нескольких лет	до нескольких лет
Легкость использования	просто	просто	сложно
Определение основных требований в начале проекта	да	нет	нет
Масштаб проекта	малые и средние	малые и средние	любые проекты
Внесение изменений	нет	да	да
Разработка итерациями	нет	да	да
Распространение промежуточного ПО	нет	да	да
Управление рисками	нет	да	да
Стоимость будущих версий	высокая	низкая	низкая
Продуктивность приложений	высокая	низкая	высокая

Согласно данной методике, наиболее приемлемой моделью жизненного цикла будет являться та, в соответствующем столбце которой будет выбрано большее число оценок. В заключении следует отметить, что каждая из представленных моделей жизненного цикла программного обеспечения имеет свои достоинства и недостатки, поэтому последовательность этапов разработки может существенно отличаться. Модели жизненного цикла информационных систем предназначены для использования, в первую очередь, разработчиками этих систем. Очень важно выбрать именно такую модель, которая будет востребована при реальной эксплуатации, в наибольшей степени отвечая характеру проекта и реальным условиям его реализации.

Литература:

Голосовский М. С. Информационно-логическая модель процесса разработки программного обеспечения // Программные системы и вычислительные методы. 2015. № 1. С. 59–68.

Ларкин Е. В., Богомолов А. В., Привалов А. Н. Методика оценивания временных интервалов между транзакциями в алгоритмах сжатия речевых сообщений // Научно-техническая информация. Серия 2: Информационные процессы и системы. 2017. № 9. С. 23–28.

Голосовский М. С. Модель расчета оценок трудоемкости и срока разработки информационных систем на начальном этапе жизненного цикла проекта // Программная инженерия. 2016. Т. 7. № 10. С. 446–455.

Жизненный цикл малого предприятия. — М.: Либеральная Миссия, Новое литературное обозрение, 2016. — 336 с.

Исаев Г. Н. Управление качеством информационных систем — М.:НИЦ ИНФРА-М, 2016. — 248 с.

Основные термины (генерируются автоматически): жизненный цикл, программное обеспечение, модель, каскадная модель, программный продукт, спиральная модель, IEC, большее число оценок, информационная система, приемлемая модель.