КОНСПЕКТ ЛЕКЦИЙ

|  |  |
| --- | --- |
| *по дисциплине* | Проектирование человеко-машинного интерфейса |
|  | (наименование дисциплины) |

Тема лекционных занятий: 7-8. Документирование и верификация требований. Управление требованиями

***Литература:***

1. IEEE Std 610.12-1990 «IEEE Standart Glossary of Software Engineering Terminology»;
2. SWEBOK «IEEE Guide to the Software Engineering Body of Knowledge».
3. ГОСТ Р ИСО 9000-2001 «Системы менеджмента качества. Основные положения и словарь»
4. Вигерс Карл, Битти Джой. Разработка требований к программному обеспечению. 3-е изд., дополненное / Пер. с англ. – М.: Издательство «Русская редакция»; СПб. БХВ-Петербург, 2014
5. Traceability with JIRA and Confluence Primer [Электронный ресурс]: YouTube - URL: <https://www.youtube.com/watch?v=ti8f2j0Pqm4>
6. Traceability Matrix and Link Graph [Электронный ресурс]: ATLASSIAN - URL: Marketplace <https://marketplace.atlassian.com/plugins/com.kostebekteknoloji.plugins.jira.jira-traceability-matrix/server/overview>
7. A Study of Semi-Automated Tracing by Jeffrey Holden, 2011
8. How to input complex degrees? UX design of a degree component in a professional software [Электронный ресурс]: UX Collective - URL: <https://medium.com/@irina1nikulina/how-to-input-complex-degrees-ux-design-of-a-degree-component-in-a-professional-software-aa91fd0df5dd>

Дополнительно:

1. Analyst Days: Ирина Сурова — Использование трассировок на практике [Электронный ресурс]: habr - URL: <https://habrahabr.ru/company/sqalab/blog/221909/>

***Краткое содержание лекций по теме:***

1. *Жизненный цикл информационной системы*
2. *Жизненный цикл требований, документирование требований. Управление требованиями*
3. *Как не сделать лишнего и сделать нужное*
   1. *Матрица доступа к данным*
   2. *Прототипирование пользовательского интерфейса*
   3. *Верификация требований к интерфейсу*
   4. *Трассировка как способ поддержки целостности в системе*

**Жизненный цикл информационной системы (ИС)**

Период создания и использования информационных систем, охватывающий совокупность состояний и этапов, которые система проходит в своем развитии от момента принятия решения о создании системы до момента прекращения функционирования системы, называется жизненным циклом ИС.

Для каждого этапа жизненного цикла определяются состав и последовательность выполняемых работ, получаемые результаты, методы и средства, необходимые для выполнения работ, роли и ответственность участников и т.д.

Такое формальное описание жизненного цикла ИС позволяет спланировать и организовать процесс коллективной разработки и обеспечить управление этим процессом.

Жизненный цикл ИС можно представить как ряд событий, происходящих с системой в процессе ее создания и использования.

Модель жизненного цикла отражает различные состояния системы, начиная с момента возникновения необходимости в данной ИС и заканчивая моментом ее полного выхода из употребления. Модель жизненного цикла - структура, содержащая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения программного продукта в течение всей жизни системы, от определения требований до завершения ее использования.

Жизненный цикл заканчивается, как правило, не в результате физического износа информационной системы, а в результате морального устаревания. Моральный износ, моральное устаревание- прекращение удовлетворения требований к информационной системе. При этом возможные модификации информационной системы экономически невыгодны или невозможны, что влечет за собой необходимость разработки новой информационной системы. Для информационных технологий является вполне естественным то, что они устаревают и заменяются новыми.

Суть содержания жизненного цикла-разработки ИС в различных подходах одинакова и сводится к выполнению следующих стадий:

**1. Планирование и анализ требований (предпроектная стадия)**

- Исследование и анализ существующей ИС

- формирование требований к новой ИС

- изучение объекта автоматизации

- выбор и разработка варианта концепции системы

- создание и утверждение технико-экономического обоснования

- разработка и утверждение технического задания на проектирование информационной системы

**2. Проектирование (техническое проектирование, логическое проектирование) выбор проектных решений по всем аспектам разработки информационной системы**

- описание всех компонентов информационной системы

- оформление и утверждение технического проекта

- выбор и разработка математических методов и алгоритмов программ

- корректировка структур баз данных

- создание документации на поставку и установку программных продуктов

- Выбор комплекса технических средств ИС

- разработка технорабочего проекта ИС

**3. Реализация (рабочее проектирование, физическое проектирование, программирование) получение и установка технических средств**

- разработка, тестирование и доводка программ

- получение и установка программных средств

- наполнение баз данных

- разработка инструкций по эксплуатации программного обеспечения, технических средств, должностных инструкций для персонала

- оформление рабочего проекта

**4. Внедрение (тестирование, опытная эксплуатация) ввод в опытную эксплуатацию технических средств**

- ввод в опытную эксплуатацию программных средств

- обучение персонала

- поэтапное внедрение ИС в эксплуатацию по подразделениям экономического объекта

- оформление акта о приемо-сдаточных испытаниях ИС

**5. Эксплуатация ИС (сопровождение, модернизация) повседневная эксплуатация ИС**

- сбор статистики о функционировании ИС

- исправление ошибок и недоработок

- оформление требований к модернизации ИС и ее выполнение.

Часто второй и третий этапы объединяют в одну стадию, называемую техно-рабочим проектированием или системным синтезом.

На каждом этапе жизненного цикла формируется набор документов и технических решений, они являются исходными для последующих решений

**Жизненный цикл требований, документирование требований. Управление требованиями**

* Выявление
* Анализ
* Документирование
* Утверждение
* Реализация
* Ревью

Постановка задач на разработку включает в себя комплект разработанных и взаимосвязанных артефактов в зависимости от задачи (разработка с нуля, доработка, обработка запроса пользователя, постановка на интеграцию), и направлена на реализацию определенной функциональности.

**Как не делать лишнего и сделать нужное [4]**

1. Организация обратной связи с пользователями при итеративной разработке;
2. Разложение требований высокого уровня на простейшие составляющие — это позволит понять, чего же именно просят пользователи. Из-за неясности требований высокого уровня, предоставляющих клиентам свободу интерпретации, возможно несовпадение представлений клиента о продукте и тем, что создает разработчик. Вот некоторые неточные и расплывчатые термины, которых следует избегать: ***обеспечивать поддержку, предоставлять возможность, разрешать, обрабатывать и управлять;***
3. Убедиться, что все классы пользователей предоставили вам информацию и для каждого варианта использования определена по крайней мере одна роль;
4. Проверка на граничные значения. Предположим, в одном требовании указано: «Если стоимость заказа меньше $100, стоимость доставки будет равна $5,95", а в другом — «Если стоимость заказа превышает $100, стоимость доставки составляет 5% от общей стоимости заказа». А как быть, если стоимость заказа составляет ровно $100? Это не оговорено, значит, отсутствует соответствующее требование;
5. Использование разнообразных форм представления информации о требованиях. Трудно прочитать большой объем текста и заметить, что чего-то не хватает. Модели анализа визуально представляют требования высокого уровня абстракции — лес, а не отдельные деревья. Рассматривая модель, вы можете заметить, что от одною блока к другому должна идти стрелка; это тоже недостающее требование. Подобного рода ошибку легче заметить на рисунке, чем в длинном списке требований, который сливается перед глазами;
6. Построение матрицы доступа к данным (см. далее);
7. Выполнение прототипирования пользовательского интерфейса (см. далее);
8. Трассируйте требования к системе, варианты использования, списки откликов на события и бизнес-правила в детализирующие их функциональные требования. Это позволит вам быть уверенным, что аналитик описал всю необходимую функциональность (см. далее).

**Матрица доступа к данным**

Один из точных способов поиска недостающих требований — создать матрицу CRUD (Create, Read, Update, Delete — создание, чтение, обновление, удаление) (см. презентацию). Она позволяет соотнести действия системы с элементами данных (отдельными или их совокупностями), в результате вы получаете представление о том, где и как каждый элемент денных создается, считывается, обновляется и удаляется. Некоторые добавляют к названию матрицы букву L указывая, что элемент данных является списком (List) (Ferdinandi, 2002). В зависимости от способов анализа требований, которые вы используете, удается исследовать различные типы соответствий, в том числе:

* элементы данных и системные события (Robertson и Robertson,1999);
* элементы данных и задачи пользователей или варианты использования (Lauesen, 2002);
* классы объектов и системные события (Ferdinandi, 2002);
* классы объектов и варианты использования (Armour и Miller, 2001).

Каждая ячейка указывает, как вариант использования, определенный в крайнем левом столбце, взаимодействует с каждым элементом данных. Вариант использования может **Создать - (С)** (Create), **Читать – (Ч)** (Read), **Обновить – (О)** (Update), **Удалить – (Уд)** (Delete) или **Указать в виде списка – (Ук)** (List) элемент данных.

Для сущностей модели предметной области в принципе будет полезно указать предпочтительный вариант представления пользователю (в зависимости от типа пользователя, функции) и мощность объекта, среднее количество экземпляров сущности в системе.

**Прототипирование пользовательского интерфейса**

Постановка задач может быть дополнена прототипами пользовательского интерфейса. Прототипы могут различаться по уровню детализации или достоверности (см. Таблица 1).

Таблица 1

|  |  |  |
| --- | --- | --- |
| Детализация (достоверность) прототипа | Преимущества | Недостатки |
| Низкая | * Меньшая стоимость разработки * Возможность оценки множества вариантов дизайнов * Представляет полезную информацию для разработчиков и дизайнеров * Решает проблемы создания макетов экрана * Может использоваться для определения потребностей рынка * Доказывает или опровергает идеи и концепции | * Ограниченный контроль ошибок * Плохая детализация спецификаций для дальнейшей разработки * Процессом управляет «посредник» * Ограниченная полезность после утверждения требований * Ограниченная пригодность для юзабилити-тестирования * Ограничения, связанные с навигацией и потоками активности |
| Высокая | * (Почти) полная функциональность * Интерактивность * Процесс разработки, управляемый пользователем * Четкая навигационная схема * Является инструментом для исследования и тестирования * Наглядно показывает конечный продукт * Служит «живой» спецификацией * Маркетинговый и торговый инструмент | * Высокая стоимость разработки * БОльшие затраты времени на создание * Не эффективен для проверки идей и концепций * Не эффективен для формирования окончательных требований |

В пользовательском интерфейсе можно условно выделить две составляющие:

* декоративную (элементы, отвечающие за эстетическую привлекательность программного изделия). Декоративная составляющая зависит от типа информационной системы, необходимой производительности, платформы реализации (системных требований);
* активную. Активные элементы подразделяются на операционные и информационные образы моделей вычислений и управляющие средства пользовательского интерфейса, посредством которых пользователь управляет программой. Управляющие средства различных классов программных изделий могут значительно различаться.

Категории требований, предъявляемых к пользовательскому интерфейсу:

* Взаимодействие интерфейса с пользователем:
  + требования к размещению элементов управления на экранных формах;
  + требования к содержанию и оформлению выводимых сообщений;
  + требования к форматам ввода;
* Взаимодействие интерфейса с внутренней логикой системы:
  + требования к реакции системы на ввод пользователя;
  + требования к времени отклика на команды пользователя;
  + требования к функциям взаимодействия и передачи информации.

**Верификация требований к интерфейсу**

1. Полнота покрытия:
   1. **функциональное покрытие** - покрытие требований к пользовательскому интерфейсу;
   2. **структурное покрытие** - для обеспечения полного структурного покрытия каждый интерфейсный элемент должен быть использован в тестовых примерах хотя бы один раз;
   3. **структурное покрытие с учетом состояния элементов интерфейса**- для обеспечения этого уровня покрытия необходимо не только использовать каждый элемент интерфейса, но и привести его во все возможные состояния (например, для чек-боксов - отмечен/не отмечен, для полей ввода - пустое/заполненное не целиком/заполненное полностью и т.п.)
   4. **структурное покрытие с учетом состояния элементов интерфейса и внутреннего состояния системы** - поведение некоторых интерфейсных элементов может изменяться в зависимости от внутреннего состояния системы. Каждое такое различимое поведение интерфейсного элемента должно быть проверено. Например, система может иметь два режима работы - нормальный и для начинающего пользователя, в котором нажатие каждого элемента сопровождается появлением всплывающей подсказки. В этом случае нужно проверить оба режима и при этом проверить, что подсказки появляются только в режиме для начинающих.
2. **Естественность (интуитивность)**. Работа с системой не должна вызывать у пользователя сложностей в поиске необходимых директив (элементов интерфейса) для управления процессом решения поставленной задачи.
3. **Непротиворечивость**. Если в процессе работы с системой пользователем были использованы некоторые приемы работы с некоторой частью системы, то в другой части системы приемы работы должны быть идентичны. Также работа с системой через интерфейс должна соответствовать установленным, привычным нормам (например, использование клавиши Enter).
4. **Неизбыточность**. Это означает, что пользователь должен вводить только минимальную информацию для работы или управления системой. Например, пользователь не должен вводить незначимые цифры (00010 вместо 10). Аналогично нельзя требовать от пользователя вводить информацию, которая уже была введена или которая может быть автоматически получена из системы. Желательно использовать значения по умолчанию, где только возможно, чтобы минимизировать процесс ввода информации [8].
5. **Непосредственный доступ к системе помощи**. В процессе работы необходимо, чтобы система обеспечивала пользователя необходимыми инструкциями. Система помощи отвечает трем основным аспектам: качество и количество обеспечиваемых команд; характер сообщений об ошибках и подтверждения того, что система делает. Сообщения об ошибках должны быть полезны и понятны пользователю.
6. **Гибкость**. Для неопытных пользователей интерфейс может быть организован как иерархическая структура меню, а для опытных пользователей как команды, комбинации нажатий клавиш и параметры.

**Трассировка как способ поддержки целостности в системе**

Процесс формирования требований к автоматизированной системе (АС) может включать в себя трассировку.

Трассировка Трассирование (Traceability) [1, 3],

Трассировка требований (Requirements Tracing) [2],

Матрица отслеживания требований (Traceability matrix) [1, 3]

*Степень, до которой можно установить связь между двумя или большим числом продуктов процесса разработки, особенно продуктами, которые являются по отношению друг к другу предшествующим-последующим или главным-подчиненным* [1].

*Степень, до которой каждый продукт процесса разработки оправдывает смысл своего существования* [1].

*Процесс определения логических связей между одним элементом системы (вариантам использования, функциональными требованиями, бизнес-правилами, компонентами дизайна, фрагментами кода, вариантами тестирования и т.д.) и другим* [3].

*Связь между требованиями и отслеживание источников требований. Трассировка предполагает направленную связь (представляется в виде сложного направленного ациклического графа) между требованиями, то есть зависимости* [2].

Назначение:

* выявление связей между объектами. По связям основополагающее это предшественник-последователь, главный-подчиненный. Позволяет взглянуть на систему (продукт) с разных сторон и точек зрения (всеобъемлюще и детально). [1,3] *вы можете использовать эти матрицы для определения различных взаимоотношений, возможными между парами требований, например «указывает/указан», «зависит от», «является родительским для» и «ограничивает/ограничен» (Sommerville и Sawyer, 1997)* [4];
* «делать то, что нужно, а не то, что можешь». Обоснование (оправдание смысла существования, [1]) объекта и поиск его источника. *Связи трассируемости помогают следить за развитием требования в обоих направлениях* *— от первоисточника к реализации и наоборот (Gotel и Finkelstein, 1994)* [4].

В каких случаях:

* неопределенность;
* невыполненные требования;
* потребность в выборе (выбор объекта для разработки\доработки);
* «что-то» документируется - разрозненность информации, либо её нехватка - «всё в голове». *Документирование взаимосвязей уменьшает риск возникновения проблем, если вдруг ключевой член команды, обладающей важной информацией о системе, покидает проект (Ambler, 1999)* [4]*.*

Применение:

* верификация связей. Матрица трассируемости требований помогает избежать пропуска каких-либо требований во время проектирования, конструирования или тестирования [4]. Способ зафиксировать опыт предыдущих лет. Собирать связи трассируемости по мере разработки не сложно (увеличение затрат на сбор и управление информации трассируемости в дальнейшем позволяет снизить общую стоимость ЖЦ продукта (неявно)), тогда как воссоздать их в завершенной системе — практически нереально;
* анализ влияния изменения (impact analysis). Трассировка является фундаментальной основой проведения анализа влияния при изменении требований, помогая предсказывать эффект от внесения таких изменений [2];
* оценка и анализ рисков. Риск - фактор, который может привести к негативным последствиям в будущем, обычно выражается через вероятность и влияние.
* регрессионное тестирование. Если тестирование дает неожиданный результат, то связи между тестами, требованиями и кодом укажут на наиболее вероятные части кода, которые необходимо проверить на наличие дефектов [4].

Инструменты и подходы:

Наиболее типичный способ представления связей между требованиями и другими элементами системами — матрица трассировки [4].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Пользовательское требование** | **Функциональное требование** | **Элемент проектирования** | **Модуль кода** | **Вариант тестирования** |
| UC-28 | catalog.query.sort | Каталог класса | catalog.sort() | search. 7  search. 8 |
| UC-29 | catalog.query.import | Каталог класса | catalog.import()  catalog.validate() | search. 12  search. 13  search. 14 |

* механизмы хранения данных:
  + JIRa (MindMap, Link, Karma, Traceability Matrix and Link Graph) [5, 6]
  + Enterprise Architect и др.;
  + электронные таблицы и др.;
* введение индивидуальных элементов трассировки для описания связей:
  + спорный вопрос, предположения\объяснения, элементы действий, термины\акронимы, библиографические ссылки
  + ХРОНИЧ - Хронический\Рискованный\Основной\Новый Измененный\Чувствительный   
    (Карен Н. Джонсон)
  + ключевые вопросыдля самопроверки при внесении изменений (зависят от организации, проекта, подхода ведения требований)
    - *Конфликтуют ли какие-то из требований (ТЕКУЩИХ\ОТЛОЖЕННЫХ) в базовой версии с предложенным изменением?*
    - *Какие могут быть последствия (ХАРАКТЕР), если изменение не будет внесено?*
    - *Выполнимо ли предложенное изменение в рамках известных технических ограничений и квалификации специалистов?*
    - *Потребуется ли создание прототипов или другая информация пользователей для проверки предложенного изменения?*
    - *Сколько затрат, уже вложенных в проект, будут потеряны, если принять это изменение?*

Последовательность действий для реализации трассирования требований:

1. Определение цели;
2. Выбор типов отношений-связей;
3. Выбор объектов системы, для которых будет поддерживаться информация трассируемости;
4. Выбор механизма хранения данных;
5. Назначение ответственного лица – координатора трассировки, и лиц, участвующих в поддержке информации по объектам системы;
6. Определение вех, по достижению которых должно происходить предоставление информации, и их механизмов.

Следует проводить проверки с целью верификации всякий раз, когда становится очевидно, что будущая, текущая или прошлая фаза внесла существенные изменения в трассировочные связи.

Заключение

1. **Back to Black**. Каждый шаг разработки должен корректный – оправдывать предыдущий шаг и удовлетворять потребности следующего шага. В любом случае при верификации важно убедиться, что ни одна связь не пропущена и все элементы более низкого уровня, такие как программные требования/прецеденты, надлежащим образом связаны с более высокоуровневыми требованиями к продукту.  
   Работу с матрицей трассировки необходимо проводить своевременно - заполнять информацию следует по мере выполнения работы, а не по мере планирования.
2. **Human is a key**. Автоматические средства трассировки позволяют проверять верификационные отношения, чтобы удостовериться, что никакие необходимые отношения не пропущены и нет лишних верификационных отношений. Сами по себе автоматические средства не могут выполнить всю работу. Верификация требует вдумчивого отношенияи нельзя полагаться на чисто механические действия [7].
3. **Без фанатизма и одержимости**. Да, скорее всего понадобится не одна таблица. Всё равно - избегать перегруза вводимых элементов (выбор правильного уровня детализации), поскольку далее поддержка трассировки становится затратной\обременительной. Узкие задачи. Следует однозначно представлять какие связи в проекте уместны.
4. **Собирать по зёрнышку**. Избегать слипания множества требований в один ком. Да, никогда не удастся создать спецификацию, в которой бы проявлялись все эти атрибуты.
5. **Инвестиции**. Рассматривайте трассируемость требований как инвестицию, увеличивающую шансы поставить продукт, который удовлетворяет все основные требования клиента. Определить связи трассируемости несложно, если вы собираете информацию по мере разработки, однако делать это для уже завершенной системы утомительно и дорого.

Разработал:

Яцык А.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.