Модели и методологии процесса разработки ПС

Модели - каскадная, спиральная, итерационная

Унифицированный процесс разработки

**Требования к ПО**

Средства автоматизированной разработки

К текущему моменту нами были выбраны 2 и 3 составляющих для выполнения успешной разработки а именно:

унифицированный процесс разработки

нотация UML

Выберем 3ю составляющую - ОО CASE-средство.

Задача

Пусть необходимо разработать ПО для теплицы работающее на гидропоника

Для нормального процесса выращивания необходимо соблюдать режим выращивания

Для этого требуется поддерживать параметры теплицы (круглосуточно)

Поддерживаемые параметры:

Температура

Уровень кислотности

Уровень влажности

Освещение

И др

Также необх: датчики и упр устройства

Опираясь на постановку задачи построим для нее диаграмму Use Case

Начинаем с оператора, который запускает процесс -> создать план выращивания

Все функции контроля и управления выполняет устройство "контроллер", который связан с протоколом работы, от протокола - на оператора. Для поддержки норм условий в системе контроллер связан с датчиком и исполнительное устройство. Датчик измеряет показатели среды и эти сведения передает контроллеру. Он анализирует, записывает в протокол работы.

**Deployment**

Моделирует аппаратную часть

После того как построена диаграмма Use Case известны все актеры и роли в системе имеет смысл построить диаграммы deployment, предназначенную для анализа аппаратной части системы. На этой диаграмме исп 2 основных элемента - processor (отображает любое устройство, выполняющее вычислительный процесс) и device (любое другое устройство)

Компьютер = контроллер и ряд датчиков и исполнительных устройств, кот потребуются в системе исходя из постановки задачи.

Диаграмма показывает, что работа системы планируется на одном компьютере, соединенном с датчиками и исполнительными устройствами. Датчиков 2 - температуры и датчик кислотности.

Исполнительные устройства: лампа, нагреватель, вентилятор, и 2 резервуара - с водой и с удобрениями.

Перед тем, как приступать к рассмотрению диаграммы классов и кодогенерации с наполнением скелета приложения методами рассмотрим проектирование на примере ещё одной задачи

Модель виртуального книжного магазина. Модель будет строится в другой CASE среде

Актеры/роли

Покупатель - любой пользователь сети, зарегистрировавшийся в магазине как покупатель

Администратор - работник, который проверяет наличие заказов пользователей, формирует их, если они есть на складе и отправляет покупателям с курьером

Директор - получает отчеты о заказанных и отправленных потребителям товарах

Функции

* Просмотр каталога - доступна всем
* Регистрация пользователя
* Работа с корзиной покупателя
* Оформление заказа на покупку
* Просмотр статуса заказа - для зарегистрированного пользователя
* Изменение статуса заказа - для администратора
* Просмотр списка заказа - директор
* Редактирование каталога продукции - администратор
* Изменение данных пользователя - администратор и зарегистрированный пользователь

Набор инструментов в данной среде расширен, чем в Rational Rose, а именно - в плане представления видов связей

Построив диаграмму Use Case проанализируем аппаратную часть, на которой будет решаться задача, для этого воспользуемся диаграммой Deployment

Диаграмма состояний:

В основу построения этих диаграмм, так же, как и в Rational Rose положен принцип работы конечных автоматов

Динамические состояния строятся при помощи диаграммы Activity большая часть - как в Rational Rose (в методе)

Диаграммы взаимодействия - в явном виде диаграмма Collaboration отсутствует

Sequence как и раньше показывает взаимодействие объектов системы во времени, можно строить разновидность диаграммы Sequence, которая отражает роли без ссылки на конкретные классы. Роль - это определенная последовательность в обмене сообщениями

Если в системе задан определенны объект, то каждая роль ссылается на базовый класс, кот идентифицируется атрибутами и операциями, нуждающимися во взаимодействии с заданным объектом

--------------------------------------

Разработка диаграммы Class

Создание интерфейсов между подсистемами

* Интерфейс *Активизировать* определяет возможность подсистемы *Управление регистрации* передавать управление системе *Управление каталогом*
* Интерфейс *Сделать заказ* определяет возможность подсистемы *Управление каталогом* передавать заказ на обработку подсистеме *Управление заказами*
* Интерфейс *Подтвердить* позволить подтверждать заказ
* Интерфейс *Послать сообщение* позволяет рассылать сообщения пользователям и администраторам

Все устройства показаны как встроенные в контроллер (связь агрегирования), так как устройства имеют схожий принцип работы, они наследуются от родительского класса устройств (generalization)

Построив диаграмму, предназначенную для кодогенерации можно генерировать код по отдельным классам или для всех классов диаграммы. При этом связи между классами также будут влиять на код. В код войдут: имена классов, методы, атрибуты. Содержание пишет разработчик

Выбрав класс для генерации получим соответствующий код на стороне среды программирования. При необходимости внесения изменения в диаграмму или код разработчик должен либо обновить код по модели, либо обновить модель по коду. Автоматически обновление не выполняется.

Структура создаваемого кода класса. Генератор кода для каждого класса строит след структуру:

1. Директивы include, которые создаются исходя из необходимости включения атрибутов и связи классов
2. Декларация, имя класса, тип и наследование
3. Атрибуты, методы
4. Служебные записи среды

Кроме генерации кода отдельных классов среда предоставляет возможность генерации шаблона целого приложения.

В качестве структуры приложения выберем простое оконное приложение с отчетом по работе теплицы

В результате использования стандартной библиотеки:

Класс просмотра

Класс документа

Основного окна

Автоматически был построен шаблон приложения

С каждым из классов свяжем какую-то часть функций

Отчет по работе АСУТП

Добавление функциональности в класс документа

Наполнение функционала класса. В нем будут хранится основные данные по устройствам и датчикам. Для этого создадим структуру **condition.** Она будет моделировать состояние среды. Также будут хранится реальные состояния исполнительных устройств. Также информация о дате и времени. До сих пор в системе нет датчиков, с которых можно считывать реальную информацию, поэтому создадим класс, который будет вносит изменение. Создадим класс окружения с операциями Change и Update. Первая - вносит изменения, вторая - фиксирует и вносит в структуру **condition**. Создадим класс Clog. Добавив все необходимые методы в эти классы, выполняем обновление кода по модели.

Для оптимизации работы системы внесем изменения в диаграмму классов, добавив родительский класс устройств и сделав дочерними все имеющиеся устройства.

Создание адреса расположения устройств. Если воспринимать каждое устройство как физическое, то каждое из них должно характеризоваться своим местом расположения. Запланировали класс Location. Создадим класс устройств из которого будут наследоваться классы исполнительных устройств и датчиков. Этот класс будет отвечать за обработку расположения устройств.

-----------------------------

**Современные технологии создания ПО**

Основные принципы:

ЖЦ

ООП  
Автоматизация

(Унифицированный подход)