# Приемы обеспечения технологичности ПО

Юнит-тесты

Под технологичностью понимают качество проекта ПО, от которого зависят трудовые и материальные затраты на его реализацию и последующую модификацию

Хороший проект сравнительно быстро и легко кодируется, тестируется, отлаживается и модифицируется

Технологичность ПО определяется следующим:

Проработанностью его моделей

Уровнем независимости модулей

Стилем программирования

Степенью повторного использования кода

Проработанность модели – чем лучше проработана модель, тем четче определены подзадачи и структуры данных, проще проектирование и меньше вероятность ошибок

Независимость моделей – чем выше независимость модулей, тем их легче понять, реализовывать, модифицировать, а также находить ошибки и их исправлять

Стиль программирования – под этим понимают стиль оформления программы, ее структурность, что существенно влияет на читаемость кода и кол-во ошибок программирования

Увеличение степени повторного использования кода, хорошо если это использование ранее разработанных библиотек, а также если это унификация кодов текущей разработки

Если же степень исходного кода повышается искусственно, то технологичность снижается.

Высокая технологичность проекта особо важна, если продукты рассчитаны на многолетнее использование

# Модули и их свойства

Модулем называется автономно компилируемая программная единица

1. Сцепление – является мерой взаимозависимости модулей, которое определяет то, насколько хорошо модули отделены друг от друга

Различают 5 видов сцепления:

1. Сцепление по данным – предполагает, что модули обмениваются данными, представленными скалярными значениями. При небольшом количестве передаваемых параметров этот тип сцепления обеспечивает наилучшие технологические характеристики ПО (самое лучшее, что может быть)
2. Сцепление по образцу – предполагает, что модули обмениваются данными, объединенными в структуры. При изменении структуры данных необходимо переписать модули, которые используют ее, а также прозрачность связи между модулями становится меньше
3. Сцепление по управлению – предполагает, когда один модуль посылает другому некоторый информационный объект, предназначенный для управления внутренней логикой модуля. Таким образом очень часто выполнят настройку режимов работы ПО.
4. Сцепление по общей области данных – предполагает, что модули работают с общей областью данных – на текущий момент такое сцепление считается недопустимым
5. Сцепление по содержимому, когда один модуль содержит обращение к внутренним компонентам другого, например, читает или изменяет внутренние данные, коды, передает управление вовнутрь второго модуля
6. Связанность модулей – мера прочности соединения функциональных и информационных объектов внутри одного модуля. Она характеризует степень взаимосвязи элементов, реализуемых одним модулем. Различают следующие виды связи (даются в порядке убывания уровня):  
   1) Функциональная связанность – когда все объекты модуля предназначены для выполнения одной функции, например, операции, объединяемые для выполнения одной функции или данные, связанные с. одной функцией

2) Последовательная связанность - Когда выход одной функции служит исходными данными для другой. Как правило такой модуль реализует одну подпрограмму, выполняющую две функции. Функциональная или последовательная связанность.

3) Информационная связанность – когда функции внутри модуля обрабатывают одни и те же данные

4) Процедурная связанность – такая связь имеет функции и данные, которые являются данными одного процесса.

5) Временная связанность – подразумевает, что функции выполняются параллельно или в течении определенного периода времени, например при инициализации (запуске) некоторого процесса.

6) Логическая связанность – основывается на объединении функции данных в одну логическую группу, например, функция обработки текстовой информации и т.д.

7) Случайная связанность – связь между элементами была или отсутствует.

# Нисходящая и восходящая разработка ПО

Восходящий подход к ПО – при использовании такого подхода сначала проектируют и реализуют компоненты нижнего уровня, а затем всех последующих и осуществляют их сборку. Исторически такой подход появился раньше.

Нисходящий подход к программированию – проектирование и последующая реализация выполняется сверху вниз, нереализованные компоненты заглушаются.

Защитное программирование (защита от ошибок) – при таком программировании необходимо проверять прежде всего:

1. Правильность операций ввода/вывода
2. Допустимость промежуточных результатов (типы, индексы, переменные и т. д.)

Принято различать следующие виды ошибок:

1. Ошибки передачи (данные искажаются аппаратными средствами)
2. Ошибки преобразования – программа неверно преобразует исходные данные во внутренний формат.
3. Ошибки перезаписи – пользователи вводят лишний или другой символ
4. Ошибки данных – пользователь вводит неверные данные

Проверка допустимости промежуточных результатов – проверяем только те промежуточные результаты, проверка которых целесообразна и не особо сложна:

* Проверка правильности вычислений индекса массива
* Значения переменных, которые определяют управляющее значение во внутренних циклах
* Предотвращение накопления погрешностей, для предотвращения необходимо:  
  1) Необходимо избегать вычитания близких чисел  
  2) Избегать деления больших чисел на малые  
  3) Сложение длинной последовательности чисел  
  4) Не использовать условие равенства вещественных чисел

Сквозной структурный контроль – реализация:

Для всех этапов генерируются списки наиболее часто встречающихся ошибок, которые формируют из опыта предыдущих разработок и различных источников информации.

Эффективными считаются программы, требующие минимального времени выполнения и минимального объема оперативной памяти.

Способы уменьшения времени выполнения:

1. Избегать использования преобразования переменных и типов
2. Избегать длинных операций умножения и деления, заменяя их сдвигами: сложением и вычитанием
3. Оптимизировать запись условных выражений, исключая лишние проверки
4. Выносить вычисления константных, т.е. не зависящих от цикла за цикл выражений (ну ты понял, да)
5. Исключать многократные обращения к элементам массивов по индексам.

Способы сэкономить память:

1. Выбирать алгоритмы обработки, не требующие дублирования исходных данных структурных типов в процессе обработки (Класс String)
2. Следует помнить, что при передаче структурных данных по значению, копии этих данных различаются в оперативной памяти, при передаче по ссылке такого нет, так что используем ее.

Обеспечение эффективности не должно требовать серьезных трудовых и временных затрат, а также приводить к существенному ухудшению технологичности.