Тассов Кирилл Леонидович

Лекция 1

Технология – наука об искусстве. Технология нужна для:

- упрощения разработки
- повышения надежности
- стандартизации процесса
- уменьшения стоимости продукта.

Для нас технология это набор приемов и методов при создании программного продукта. Методология – набор приемов и методов, объединенных общей, философской идеей.

Технология должна чем-то поддерживаться. Поэтому под технологию создаются языки программирования.
Идеи языков:

- Структурные языки (например, C).
- Процедурные языки – разбиваем задачу на подзадачи.
- Декларативный подход – языки “искусственного интеллекта”. Сейчас почти не используется. Идея в выделении набора правил и законов, на которые потом накладываются задача. При этом как пойдет вычислительный процесс неизвестно. (как юрист + свод законов + проблема). Здесь основной проблемой будет решение сложных задач, так как идеальных правил не бывает.
- Функциональные языки – выделение функций, как в математике (работа не с данными, а с набором функций)
- Объектно-ориентированный подход

Структурное программирование
(программирование черного ящика, программирование без goto)

В 50 годы прошлого столетия начинает возрастать мощность вычислительной техники, появляются языки программирования высокого уровня(?). Возникает потребность в написании больших программ.

Возникшие проблемы – непонятны ресурсы, непонятно время разработки, низкая надежность, код кака.

В этот момент (1961 год) появляется идея структурного программирования (называлась она правда тогда не так).

Выделили три основные идеи:

1. Нисходящая разработка
2. Структурный контроль
3. Использование базовых логических структур

Этапы разработки:

1. Анализ (оценка задачи, которая перед нами поставлена, формулировка тз)
2. Проектирование
3. Кодирование
4. Тестирование

Во 2-4 этапе используется нисходящий подход. Разработка идет сверху вниз, поэтому логика идет снизу вверх, данные сверху вниз.

Задача разбивается на подзадачи.
Создается “планировщик”, который знает, в каком порядке, что выполнять. Подзадачи закрываются заглушками, возвращающие определенные данные. На этом этапе создаются тесты. Отладив верхний модуль, мы избавимся от самых серьезных логических ошибок.

Далее, используя идею черного ящика, которая состоит в том, что нам не важно как реализована подзадача, важно только что на входе и на выходе, мы реализуем подзадачи. При этом их можно разбивать внутри также, как мы разбили основной процесс выше.

При таком подходе у нас обеспечивается комплексное тестирование. То есть для каждой функции отдельную отладочную программу писать не нужно будет (так как на уровень выше функция уже отлажена и можно тестить их вместе).

ПРАВИЛО: каждый думает сам за себя. Т.е. реализовывая задачу, мы не думаем о подзадачах. Это обеспечивает дает надежность программного продукта.

ПРАВИЛО: все переданные данные должны быть явными, пришедшими через список параметров. Вытекает из того, что принимая данные функция должна проверить пришедшие ей данные (т.е. … не прокатят, так как мы не можем их проверить, видимо).

ПРАВИЛО*: мы не столько пишем программу, сколько ее читаем, поэтому код должен быть простой. *На код вам должно быть приятно смотреть, но не перегните палку – возбуждаться от кода не надо.*

ПРАВИЛО: структурируем данные так, чтобы передавать только то, что нужно и ничего лишнего. С полями полей структур мы не работаем. Модули должны быть максимально независимы по данным.

Можно заметить, что в этом подходе мы объединяем проектирование, тестирование и кодирование. Эти этапы объединены.

Как формируется технического задание?
ТЗ заказчика это кака. Мы его анализируем, пишем свое тз, отдаем обратно заказчику – начинается итерационный процесс. В итоге очень важно, чтобы все было формализовано до мелочей.
Момент усталости – момент, когда тз уже выучено наизусть и взгляд замыливается.

В 90 годы, когда развалился союз, люди начали заниматься тремя вещами: бандитизмом, проституцией и программированием.

Из того, что мы разрабатываем программу сверху вниз, мы получаем то преимущество, что заказчик видит как мы работаем и может быстро реагировать на ошибки в нашей разработке.

ПРАВИЛО: зависимая подзадача не должна принимать решения за модуль более высокого уровня. Она возвращает управление и там принимается решение. (то есть ни exit, ни abort не используется).

Для каждого действия должна быть дана простая формулировка. Эта формулировка не должна включаться в себя перечисления (функция делает одну вещь, а не несколько) и для этой формулировки не должно быть похожих (одну должность не занимают несколько человек)

Смещение уровней абстракции – вынос функции из функции (??). Типа такое лучше не делать.

ПРАВИЛО: зависимых подзадач не должно быть больше 7. Если больше, то надежность падает.

ПРАВИЛО: ограничиваем сложность каждой функции. 20-30 строк. Функция разбивается на логические куски – сегменты/другие функции (лучше, если это будут функции).

ПРАВИЛО: использование базовых логических структур. Так как код не столько пишется, сколько читается, то количество этих конструкций необходимо свести к достаточному минимуму.
Фирма IBM выделила сл конструкции: if (две альтернативы) и switch (много альтернатив). Глубина их вложенности конструкций должна быть < 3.
Также были выделены 4 конструкции цикла: while, until, for, loop (loop – выход один, и он находится внутри тела цикла). Из цикла должен быть только один выход.

Других конструкций нет! Избыточность это зло.

Выделение конструкций дает:
- просто читать программу
- просто переходить на другой язык
- можно читать программу на незнакомом языке

Сквозной структурный контроль.
Руководитель не участвует в написании кода, он только распределяет задачи между командами программистов и контролирует их выполнение. Из-за этого он теряет навык и *деградирует*. Для того, чтобы решить эту проблему убираем функцию контроля. Эта функция переносится на самих программистов.