Программная инженерия

Кодирование (разработка) ПО – 1

Software construction

Вопросы

- SWEBOK KA-3
- Условия реализации проекта
- Организационные структуры
- Планирование разработки, распределение задач
- Контроль качества
  - *Стандарты программирования*
  - Метрики для исходного кода

Связь с другими КА

Основные принципы

Управление разработкой

SWEBOK: KA-3: SOFTWARE CONSTRUCTION

КА-3: Software construction

- Создание ПО – процесс создания работающего ПО путем комбинирования процессов:
  - Кодирования
  - Верификации
  - Модульного тестирования
  - Интеграционного тестирования
  - Отладки

Связь КА-3 с другими

- «Конструирование ПО» это нечто между:
  - «проектированием» (КА-2) и
  - «тестированием» (КА-4),

  сильно связанное с тем и другим
- «Конструирование ПО» порождает б***о***льшую часть элементов, которые подлежат «конфигурированию» (КА-6)

Минимизация сложности

Возможность изменений

Создание верифицируемого кода

Повторное использование

Стандарты разработки

SWEBOK: KA-3.1: ОСНОВНЫЕ ПРИНЦИПЫ

Основные принципы

- Основные принципы построения ПО:
  - Минимизация сложности (minimizing complexity)
  - Возможность изменений (anticipating change)
  - Создание верифицируемого кода (constructing for verification)
  - Повторное использование [кода] (reuse)
  - [Использование] стандартов разработки (standards in construction)
- Первые 4 принципа применимы также и к проектированию ПО

Минимизация сложности – 1/2

- 7 ± 2 → минимизация сложности: «Лучше проще, чем умнее»
- Проявляется при:
  - Проектировании: создание структур кода и данных
  - Изменении ПО: анализ и собственно изменение
  - Тестировании: анализ, построение [полного покрытия] проверок, проведение тестов

Минимизация сложности – 2/2

- Достигается:
  - Использованием стандартов
  - Модульной архитектурой
  - Внедрением специальных языковых конструкций
  - Специальными техниками обеспечения качества кода

Возможность изменений

- Длинный жизненный цикл → Необходимость изменений
- Быстро меняющаяся среда и/или пользователи → Необходимость изменений
- Возможность изменений состоит в том, что создается расширяемое ПО. То есть такое, что есть возможность его улучшить без внесения помех в [текущую] работу
- Реализация возможности изменений поддерживается многими из техник кодирования

Создание верифицируемого кода – 1/2

- Создание верифицируемого кода означает построение ПО таким образом, чтобы ошибки было легко обнаруживать:
  - Непосредственно разработчикам в процессе кодирования
  - Тестировщикам при тестировании разного рода
  - Конечным пользователям в процессе независимого тестирования или использования

Создание верифицируемого кода – 2/2

- Для создания верифицируемого кода используют стандарты программирования:
  - Задающие возможность анализа кода (code reviews)
  - Вводящие модульное тестирование
  - Описывающие подготовку кода к автоматическому тестированию
  - Ограничивающие использование сложных (синтаксически) или трудных для понимания языковых конструкций

Повторное использование – 1/3

- Общее определение: Reuse это использование существующих элементов (assets) при решении **различных** задач
- В разработке ПО повторно используемые элементы это: библиотеки, модули, компоненты, исходные коды, [коммерческие] продукты (COTS)

Повторное использование – 2/3

- Повторное использование лучше использовать (is best practiced) систематически, в соответствии с хорошо определенной (well-defined) повторяемой процедурой. В таком случае оно заметно:
  - Увеличивает скорость работы
  - Повышает качество ПО
  - Уменьшает цену разработки

Повторное использование – 3/3

- Повторное использование включает:
  - «разработку для …» – создание повторно используемых элементов
  - «разработку с использованием» – собственно повторное использование элементов
- Повторное использование часто преодолевает рамки проектов. При этом повторно используемые элементы могут изначально быть созданы для других проектов и/или организаций

Стандарты разработки – 1/5

- Применение внешних (общих) или внутренних стандартов разработки ПО помогает достичь заданных в проекте показателей эффективности, качества и стоимости
- Например, определение разрешенных подмножеств ЯП и использование стандартов помогает повысить безопасность получаемого кода (с точки зрения наличия ошибок и/или информационной безопасности)

Стандарты разработки – 2/5

- Что стандартизуется? → То, что влияет на разработку напрямую:
  - Способы описания / обмена информацией (между разработчиками)
    - Например: форматы и содержание документов (и список типовых документов, и форматы диаграмм…)
  - ЯП
    - Например, ISO/IEC 14882:2017 определяет C++17
  - Стандарты программирования (coding)
    - Оформление кода: правила именования, форматирования, табуляции…
    - Использование конструкций: не использовать **`goto`**, использовать только **`shared_ptr`**…

Стандарты разработки – 3/5

- Области стандартизации (продолжение) :
  - Платформы
    - Например: Windows API, POSIX
  - Утилиты
    - Например, нотации описания типа UML

Стандарты разработки – 4/5

- Внешние стандарты
  - Описывают «крупные», заимствованные элементы: ЯП, утилиты разработки, интерфейсы. В целом задают связь между действиями, описанными в SWEBOK KA-3 и в других разделах
  - Могут быть разного уровня и статуса:
    - Международные межгосударственные всемирные (ISO, IEC, ICAO), международные региональные или групповые (ЕС, СНГ, NATO), государственные (ГОСТ, DIN), отраслевые
    - Международные негосударственные (OMG, IEEE)
    - Промышленные корпоративные

Стандарты разработки – 5/5

- Внутренние стандарты
  - Создаются организацией или кооперацией для конкретного проекта или упорядочения работ
  - Обычно описывают роли и процессы взаимодействия при групповой разработке и других этапах ЖЦ проектов
  - Предназначены для (см. список принципов): уменьшения сложности (организационной и технической), обеспечения реализации ожидаемых изменений (anticipating change), обеспечения верифицируемости продукта (constructing for verification)

Место разработки в ЖЦ

Планирование

Оценка

SWEBOK: KA-3.2: УПРАВЛЕНИЕ РАЗРАБОТКОЙ

Разработка и ЖЦ – 1/4

- Существует множество моделей ЖЦ ПО. В зависимости от назначения они по-разному концентрируются на разработке
- Линейные и близкие к ним модели ЖЦ («водопадная» модель или модель поэтапной разработки (staged-delivery)) рассматривают разработку как деятельность, совершаемую после серьезной подготовительной работы:
  - Детализации ТЗ
  - Подробного проектирования
  - Детального планирования

Разработка и ЖЦ – 2/4

- Чем «линейнее» подход, тем больше в нем подчеркивается, что подготовительные работы (работа с требованиями и проектирование) отделены от разработки. В таких моделях под разработкой понимают, в основном, кодирование

Разработка и ЖЦ – 3/4

- Другие, более итеративные модели ЖЦ (эволюционного прототипирования и гибкой разработки) рассматривают разработку как выполняемую параллельно с другими работами (включая работу с требованиями, проектирование и планирование) или пересекающуюся с ними. В таких подходах смешивают проектирование, кодирование и тестирование, рассматривая из вместе как разработку

Разработка и ЖЦ – 4/4

- Таким образом, точное определение того, что понимается под «конструированием» зависит от используемой модели ЖЦ
- Будем обобщенно считать, что в конструирование входят:
  - Кодирование и отладка (основное)
  - Планирование конструирования
  - Детальное проектирование
  - Модульное и интеграционное тестирование

Планирование разработки – 1/2

- Планирование разработки определяет метод разработки, который влияет на:
  - «Край» (сроки или объем), до которого выполняются процессы, предшествующие конструированию
  - Последовательность, в которой они выполняются
  - Степень их завершения до начала конструирования

Планирование разработки – 2/2

- Планирование также определяет:
  - Последовательность создания и интеграции компонентов
  - «Стратегия» интеграции (например, инкрементная или по фазам)
  - Процессы управления качеством ПО
  - Как распределять задачи по разработчикам (исполнителям) [software engineers]
  - Другие задачи согласно выбранному методу

Измерения – 1/2

- Что можно измерить: действия (activities) и их результаты. Например:
  - [Объём кода] разработанного, измененного, повторно использованного, удаленного
  - Сложность кода
  - Статистику исследования (inspection) кода
  - Потоки [найденных] ошибок и исправлений
  - Затраты (трудозатраты) (effort)
  - [Соблюдение] графика (scheduling)

Измерения – 2/2

- Результаты измерений могут использоваться для управления, управления качеством, модификации процессов управления (см. KA-8: Software Engineering Process)

Проект

Языки

Кодирование

Тестирование

Для/с повторного использования

Качество

Интеграция

SWEBOK: KA-3.3: ПРАКТИЧЕСКИЕ ЗАМЕЧАНИЯ

Условия

Их взаимосвязь и свойства

Факторы успеха

Ограничения реализации

УСЛОВИЯ РЕАЛИЗАЦИИ ПРОЕКТА

Условия реализации проекта – 1/2

- Персонал:
  - Профессионализм
  - Сработанность
  - Стабильность
  - Мотивация
  - Эффективность коммуникаций
- Продукт:
  - Техническая сложность (новый продукт, новые технологии, инновации)
  - Критичность для заказчика (угроза жизни, денежные потери)

Условия реализации проекта – 2/2

- Проект:
  - Большой
  - Средний
  - Малый
- Обеспечение:
  - Финансовое
  - Организационное
  - Правовое

Взаимосвязь и динамичность – 1/2

- Условия реализации проекта динамичны:
  - Команда:
    - Проходят стадии формирования команды
    - Команда изменяется количественно (как правило, растет)
    - Команда изменяется качественно (обычно, улучшается)
  - Продукт (sic!)
    - Изменение условий использования (ну, ладно…)
    - Изменение технических условий (куда ж без этого…)
    - Изменение требований (…а что делать?)
  - Обеспечение (обычно, ухудшается -_-)

Взаимосвязь и динамичность – 2/2

- Условия реализации проекта взаимосвязаны:
  - Сложность проекта сильно зависит от характеристик команды
  - Обеспечение проекта зависит от отношения заказчика
  - Отношение заказчика может сильно зависеть от команды

→ Процесс выполнения проекта должен постоянно адаптироваться к этим изменениям

«Железный треугольник»

Ограничения проекта

Факторы успеха проекта

- Чтобы программный проект стал успешным, необходимо:
  - Четко ставить цели
  - Динамичность:
    - Определять способ достижения целей
    - Контролировать и управлять реализацией
    - Анализировать угрозы и противодействовать им
    - Создавать команду

То же + мотивация

- Четыре фактора успеха (проект должен быть):
  - Выполнен в соответствие со спецификациями
  - Выполнен в срок.
  - Выполнен в пределах бюджета.
  - Каждый участник команды уходил с работы в 18:00 с чувством успеха.

Функциональная

Проектная

Матричная

ОРГАНИЗАЦИОННЫЕ СТРУКТУРЫ ПРИ РАЗРАБОТКЕ ПО

Организационные структуры

- Организационная структура компании отражает ее внутреннее устройство, потоки управляющих воздействий, распределение труда и специфические особенности производства. Обычно выделяют:
  - Функциональная организация
  - Проектная организация
  - Матричная организация
- Нет одной лучшей организационной структуры. Нет смысла противопоставлять функциональные структуры и проектные организации

Функциональная (иерархическая) структура – 1/3

Функциональная структура: за и против – 2/3

- Преимущества:
  - Сохраняется принцип единоначалия
  - Понятные и стабильные условия работы
  - Хорошо приспособлена для операционной деятельности
  - Специализация подразделений позволяет накапливать экспертный опыт

Функциональная структура: за и против – 3/3

- Недостатки:
  - Затруднено принятие решений и коммуникации между исполнителями. Осуществляются только через руководство
  - Управление сконцентрировано и держится на компетенции высшего руководства
  - Как правило, неэффективен контроль за ходом проекта (нет целостной картины)

Проектная структура – 1/2

Проектная структура: особенности – 2/2

- Особенности:
  - Проект организуется как самостоятельное производственное подразделение
  - Персонал на проект набирается по временным контрактам
  - После завершения проекта персонал увольняется
- Следствия:
  - Медленный старт проекта
  - Опыт не аккумулируется
  - Команды не сохраняются

Матричная структура – 1/5

Слабая матрица

Матричная структура – 2/5

Сбалансированная матрица

Матричная структура – 3/5

Сильная матрица

Матричная структура: за и против – 4/5

- Преимущества:
  - Информация по конкретному проекту есть в полной мере у менеджера
  - Понятные и стабильные условия работы исполнителей
  - Хорошо приспособлена для операционной деятельности
  - Специализация подразделений позволяет накапливать экспертный опыт

Матричная структура: за и против – 5/5

- Недостатки:
  - У каждого работка, как минимум два начальника
  - Сложнее сформулировать и отслеживать признаки качества работы
  - В «слабом» и «сбалансированном» вариантах статус руководителю проекта неочевиден
  - В состоянии перегруженности конфликтуют также и руководители проектов

Work Breakdown Structure

Примеры

Практики

ДЕТАЛИЗАЦИЯ ОБЪЕМА РАБОТ

Детализация объема работ

- Иерархическая структура работ (ИСР) (Work Breakdown Structure, WBS) – ориентированная на результат иерархическая декомпозиция работ, выполняемых командой проекта для достижения целей проекта и необходимых результатов
  - С ее помощью структурируется и определяется все содержание проекта
  - Каждый следующий уровень иерархии отражает более детальное определение элементов проекта

Пример декомпозиции: ГОСТ

- Например, ГОСТ 19.102-77 предусматривает каскадный подход и определяет следующие стадии разработки ПС:
  - Техническое задание
  - Эскизный проект
  - Технический проект
  - Рабочий проект
  - Внедрение
- Внезапно! Готов верхний уровень разбиения

Пример декомпозиции: Модульность

- Проект
  - Продукт 1
    - Компоненты продукта 1
      - Функции компонента
  - Продукт 2
  - …
- Внезапно! Вполне рабочая декомпозиция

Декомпозиция: tips & tricks – 1/5

- Обычно два приведенных выше подхода совмещают в единой структуре разбиения
  - Первый подход задает ограничения (обычно внешние «реперные точки» сдачи проекта и его оплаты в том числе NB!)
  - Второй подход позволяет планировать «внутренности» и отслеживать продвижение разработки

Декомпозиция: tips & tricks – 2/5

- Учитывать ресурсы:
  - Персонал
  - Оборудование (для разработки, для сопряжения, тестовые платформы)
  - Исходные данные (для тестирования)
- Учитывать время:
  - Сроки проекта
  - Другие проекты (свои и смежников)
  - Отпуска и праздники

Декомпозиция: tips & tricks – 3/5

- Параллельность и последовательность
  - Для ускорения планируют выполнение некоторых задач одновременно
  - Возможность начала некоторых задач зависит от начала или завершения других
- Принцип дробления:
  - Для лучшей отслеживаемости и надежности задачи дробят на более мелкие
  - Часто зависимости между крупными задачами можно перенести на более низкий уровень подзадач, что повышает параллельность

Декомпозиция: tips & tricks – 4/5

- Пример дробления:
  - Задача «Разработать пользовательский интерфейс» разбивается на:
    - Определение API модуля
    - Разработку форм графического интерфейса
    - Реализацию поведения GUI
    - Тестирование и отладку
  - Тогда:
    - Начать интеграцию с другими компонентами можно сразу после определения API
    - Формы графического интерфейса могут быть разработаны отдельными специалистами
    - Тестирование и отладку можно совместить с разработкой основной функциональности других компонентов

Декомпозиция: tips & tricks – 5/5

- Принцип итераций:
  - Разработка некоторых компонентов повторяется в проекте несколько раз, обеспечивая «притирку» интерфейсов компонентов друг к другу
  - Обычно вводят draft/generic/final стадии разработки компонента

Стандарты программирования

Метрики

КОНТРОЛЬ КАЧЕСТВА ПО

Зачем измерять качество ПО? - Теоретически

- Определение качества существующего продукта или процесса
- Прогнозирование качества продукта / процесса
- Улучшение качества продукта / процесса

Зачем измерять качество ПО? - Практически

- Оценка стоимости и времени выполнения будущих проектов
- Оценка производительности применения новых средств и методов
- Определение тенденций производительности с течением времени
- Улучшение качества программного обеспечения
- Прогноз будущих потребностей в персонале
- Предвидеть сокращение будущих потребностей в техническом обслуживании

Мера и метрика

- Мера – количественный показатель степени, количества, или размеров некоторых атрибутов продукта или процесса
  - Например, количество ошибок
- Метрика – количественная мера, позволяющая оценить, в какой степени система, компоненты или процесс обладают заданным атрибутом. «Предположение о значении данного атрибута»
  - Например, количество обнаруженных ошибок на затраченный человеко-час

Метрики для ПО

- Метрики ПО разделяются на три основные группы:
  - Метрики размера (исходного кода программ)
  - Метрики сложности (потока управления ПО)
  - Метрики сложности потока данных программ

РАЗМЕРНО-ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Размерно-ориентированные метрики

- Общие трудозатраты: человеко-месяцы, реже человеко-часы
- Объем программы: тысячи строк исходного кода (LOC – Lines Of Code, KLOC, MLOC)
- Стоимость разработки
- Объем документации
- Количество ошибок, обнаруженных в течение года эксплуатации
- Количество людей, работавших над проектом
- Срок разработки

Количество строк кода

- «Физические» строки кода (LOC) – общее число строк исходного кода, включая комментарии и пустые строки
- «Логические» строки кода (LSI – Lines of Source Instructions) – количество команд ЯП
- Соотношение между LOC и LSI зависит от:
  - Конкретного ЯП
  - Используемых правил оформления кода

Недостатки размерных метрик – 1/2

- Неправильно сводить оценку работы человека к нескольким числовым параметрам
  - Например, менеджер проекта может назначить наиболее талантливых программистов на сложнейший участок работы
- Метрика не учитывает опыт сотрудников и их другие качества

Недостатки размерных метрик – 2/2

- Искажение: процесс измерения может быть искажён за счёт того, что сотрудники знают об измеряемых показателях и стремятся оптимизировать эти показатели, а не свою работу
- Неточность: нет метрик, которые были бы одновременно и значимы и достаточно точны
  - Например: Количество строк кода — только число строк. Не даёт представления о сложности решаемой проблемы

Преимущества размерных метрик

- Легкость использования:
  - Подсчета
  - Демонстрации
  - Автоматизации контроля
- Просто перевести в экономические значения
  - Например: Стоимость разработки = Количество человеко-месяцев * Стоимость человеко-месяца

Примеры

Размер проекта (LOC)	Плотность ошибок (ошибок / 1K LOC)
< 2K	0-25
2K – 16K	0-40
16K – 64K	0,5 – 50
64K – 512K	2 – 70
512K+	4 – 100

Примеры

Размер проекта (LOC)	Строк кода на человека в год
1K	2500 – 25000 (4000)
10K	2000 – 25000 (3200)
100K	1000 – 20000 (2600)
1000K	700 – 10000 (2000)
10000K	300 – 5000 (1600)

МЕТРИКИ СЛОЖНОСТИ

Метрики сложности

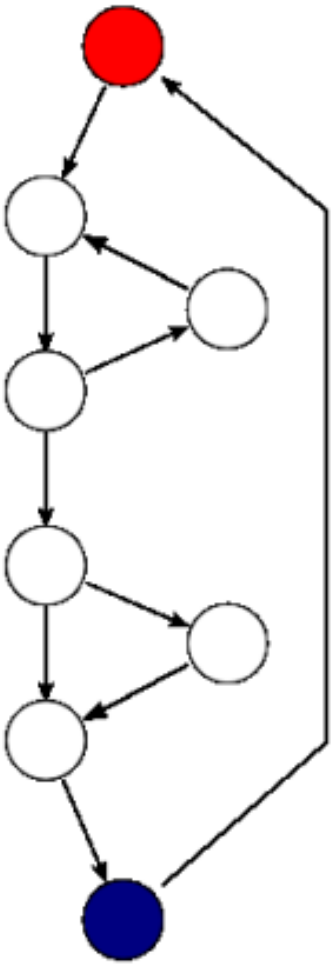
- Цикломатическая сложность: один из наиболее распространенных показателей оценки сложности программных проектов на основе графа управляющей логики
- Объектно-ориентированные: оценка сложности объектно-ориентированных проектов
- Метрики Холстеда: вычисляются на основании анализа числа строк и синтаксических элементов исходного кода программы
- Метрика Чепина: оценка информационной прочности

Цикломатическая сложность – 1/2

- Цикломатическая сложность части программного кода — счётное число линейно независимых маршрутов через программный код
- Если исходный код не содержит никаких точек решений (таких, как условия IF или циклы FOR), то цикломатическая сложность равна единице, поскольку есть единственный маршрут через код

ЦС – граф потока управления

- Графом потока управления программы называется ориентированный граф, в узлах которого находятся неделимые группы команд, а ребрами соединяются такие блоки, которые могут быть выполнены сразу друг за другом

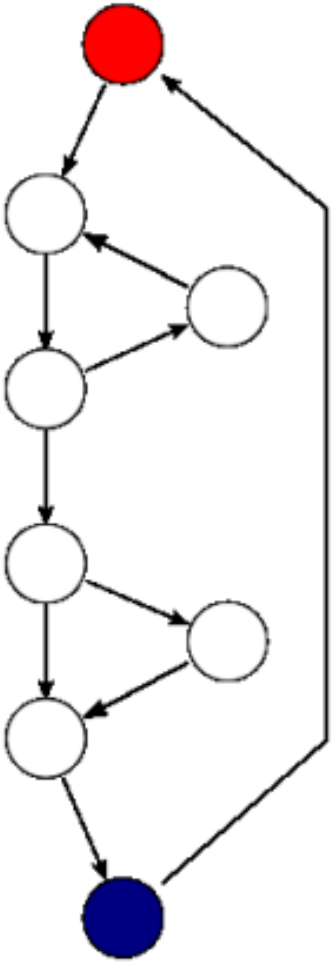


Цикломатическая сложность – 2/2

- **Показатель цикломатической сложности** позволяет не только произвести оценку трудоемкости реализации отдельных элементов программного проекта и скорректировать общие показатели оценки длительности и стоимости проекта, но и оценить связанные риски и принять необходимые управленческие решения
- $C = e - n + 2p$, где
  - e – число ребер,
  - n – число узлов,
  - p – число компонентов связности на графе управляющей логики

ЦС – пример расчета

- Программа начинает выполняться с красного узла, затем идут циклы (после красного узла идут две группы по три узла). Выход из цикла осуществляется через условный оператор (нижняя группа узлов) и конечный выход из программы в синем узле.
- Для этого графа:
  - E = 9, N = 8 и P = 1
  - Цикломатическая сложность программы равна 3



Объектно-ориентированные метрики – 1/3

Метрика	Описание
Взвешенная насыщенность класса 1 (Weighted Methods Per Class – WMC)	Отражает относительную меру сложности класса на основе цикломатической сложности каждого его метода. Класс с более сложными методами и большим количеством методов считается более сложным. При вычислении метрики родительские классы не учитываются
Взвешенная насыщенность класса 2 (Weighted Methods Per Class – WMC2)	Мера сложности класса, основанная на том, что класс с большим числом методов, является более сложным, и что метод с большим количеством параметров также является более сложным. При вычислении метрики родительские классы не учитываются

Объектно-ориентированные метрики – 2/3

Метрика	Описание
Глубина дерева наследования (Depth of inheritance tree)	Длина самого длинного пути наследования, заканчивающегося на данном модуле. Чем глубже дерево наследования модуля, тем может оказаться сложнее предсказать его поведение. С другой стороны, увеличение глубины даёт больший потенциал повторного использования данным модулем поведения, определённого для классов-предков
Количество детей (Number of children)	Число модулей, непосредственно наследующих данный модуль. Большие значения этой метрики указывают на широкие возможности повторного использования. При этом слишком большое значение может свидетельствовать о плохо выбранной абстракции

Объектно-ориентированные метрики – 3/3

Метрика	Описание
Связность объектов (Coupling between objects)	Количество модулей, связанных с данным модулем в роли клиента или поставщика. Чрезмерная связность говорит о слабости модульной инкапсуляции и может препятствовать повторному использованию кода
Отклик на класс (Response For Class)	Количество методов, которые могут вызываться экземплярами класса. Вычисляется как сумма количества локальных методов, так и количества удаленных методов

Метрики Холстеда

- Основу метрики Холстеда составляют четыре измеряемые характеристики программы:
  - NUOprtr (Number of Unique Operators) — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов)
  - NUOprnd (Number of Unique Operands) — число уникальных операндов программы (словарь операндов)
  - Noprtr (Number of Operators) — общее число операторов в программе
  - Noprnd (Number of Operands) — общее число операндов в программе

Метрики Холстеда – операнд

- Операнд — это сущность, с которой оператор выполняет какие-либо действия. Выражение — это последовательность операторов и операндов, выполняющая действия ниже в любой комбинации:
  - Вычисление значения
  - Назначение объекта или функции
  - Создание побочных эффектов
- Операнды в C включают константы, идентификаторы, строки, вызовы функций, выражения индекса, выражения выделения членов и сложные выражения, созданные путем сочетания операндов с операторами или заключения операндов в скобки
- Источник: http://msdn.microsoft.com (Операнды и выражения)

Оценки по метрике Холстеда

- Словарь программы (Halstead Program Vocabulary):
  - HPVoc = NUOprtr + NUOprnd
- Длина программы (Halstead Program Length):
  - HPLen = Noprtr + Noprnd
- Объем программы (Halstead Program Volume):
  - HPVol = HPLen log2 HPVoc
- Сложность программы (Halstead Difficulty, HDiff):
  - Hdiff = (NUOprtr/2) × (NOprnd / NUOprnd)
- На основе показателя HDiff предлагается оценивать усилия программиста при разработке при помощи показателя HEff (Halstead Effort):
  - HEff = HDiff × HPVol

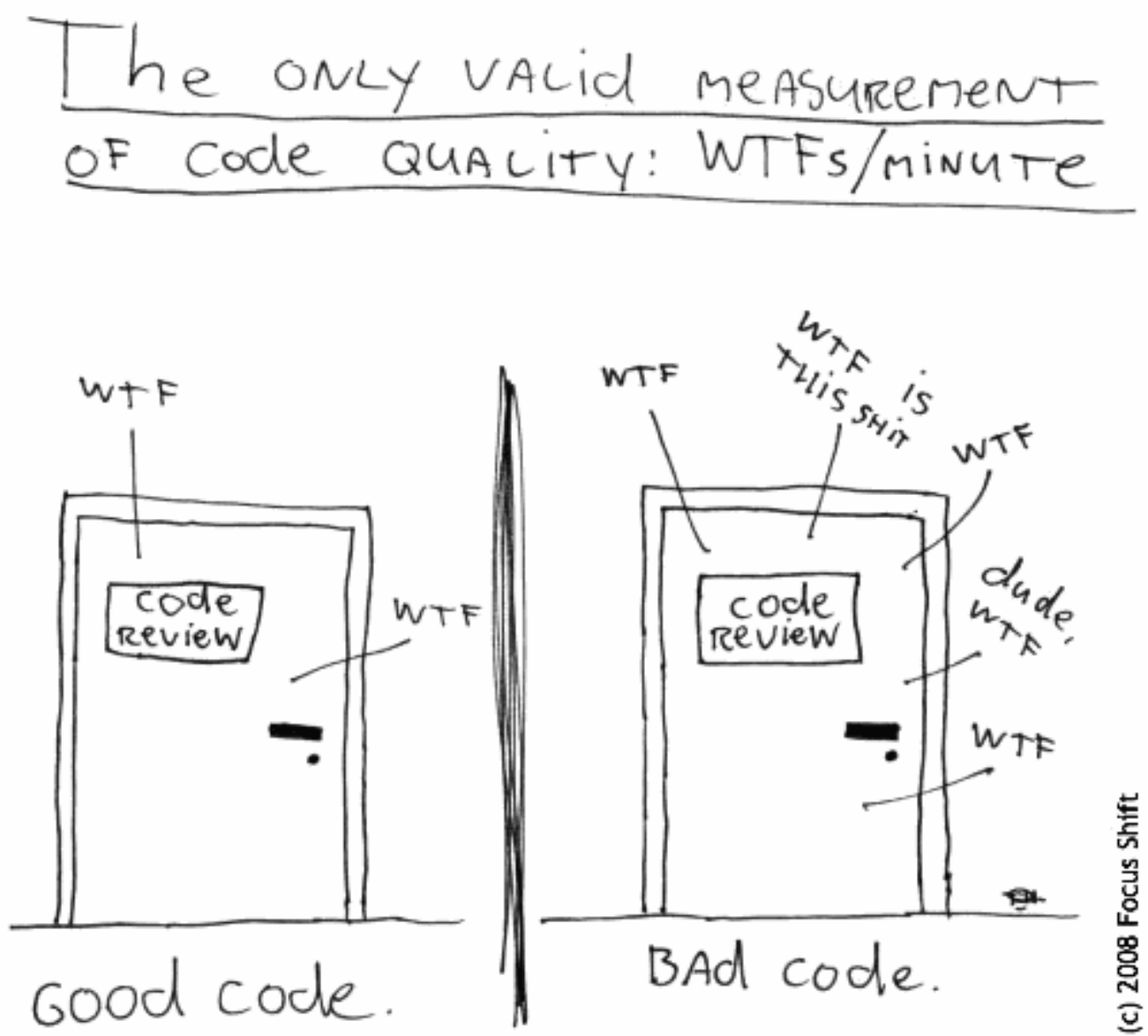
Метрики Чепина – 1/2

- Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы:
  - Множество «P» – вводимые переменные для расчетов и для обеспечения вывода. Примером может служить используемая в программах лексического анализа переменная, содержащая строку исходного текста программы, то есть сама переменная не модифицируется, а только содержит исходную информацию
  - Множество «M» – модифицируемые или создаваемые внутри программы переменные
  - Множество «C» – переменные, участвующие в управлении работой программного модуля (управляющие переменные)
  - Множество «T» – не используемые в программе («паразитные») переменные
- Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе

Метрики Чепина – 2/2

- В общем виде метрика равна:
  - Q = a1P + a2M + a3C + a4T,
  - Где a1, a2, a3, a4 – весовые коэффициенты, назначаемые специалистом

Применимая метрика



Все пропало

Ужас-ужас

ПРАКТИЧЕСКИЕ ЗАДАНИЯ