Системное программное обеспечение локальных компьютерных сетей

Введение

Денис Пынькин

2013 – 2014

e-mail: denis.pynkin@bsuir.by

http://goo.gl/32cTB

СЧАСТЬЕ ДЛЯ ВСЕХ, ДАРОМ, И ПУСТЬ НИКТО НЕ УЙДЕТ ОБИЖЕННЫЙ!

(c)Стругацкие, Пикник на обочине

Цель дисциплины

Всестороннее изучение основных вопросов, связанных с функционированием сетевого программного обеспечения компьютерных сетей для различных архитектур и операционных систем.

Задачи дисциплины

Подготовить специалиста в области сетевых технологий, разбирающегося в принципах работы и умеющего создавать системное и прикладное сетевое программное обеспечение.

Надо будет знать

- основные возможности сетевых операционных систем
- основные протоколы обмена и интерфейсы, используемые при построении глобальных и корпоративных компьютерных сетей
- области применения, достоинства и недостатки наиболее распространенных сетевых протоколов
- наиболее распространенные методы и алгоритмы взаимодействия программного обеспечения в компьютерных сетях
- принципы построения сетевого программного обеспечения
- особенности и принципы построения распределенных систем

Какого цвета учебник?

- Стивенс У. Р., Феннер Б., Рудофф Э.М., UNIX: разработка сетевых приложений. 3-е изд. – СПб.:Питер, 2007, 1039 с.
- Тенненбаум Э., Ван Стеен М., Распределенные системы. Принципы и парадигмы, Спб.:Питер, 2003, 877 с.
- Семенов Ю.А., Телекоммуникационные технологии [Электронный ресурс]. – Электронные данные. – Режим доступа: http://book.itep.ru/
- Исходники слайдов по курсу СПО ЛКС [Электронный ресурс]. – Электронные данные. – Режим доступа: https://github.com/d4s/spolks-lectures

Какого цвета учебник?

- Тенненбаум Э., Компьютерные сети, СПб.:Питер, 2003, 992 с.
- Под ред. Садыхова Р.Х., Средства параллельного программирования в ОС Linux: Учебное пособие, Мн.: ЕГУ, 2004, 476 с.
- Камер Д.Э., Стивенс Д.Л., Сети TCP/IP, том 3. Разработка приложений типа клиент/сервер для Linux/POSIX, М.:Издательский дом "Вильямс 2002, 592 с.
- Скляров И.С., Программирование боевого софта под Linux, БХВ-Петербург, 2007, 416 с.
- Реймонд Эрик С., Искусство программирования для UNIX.: Пер. С англ. – М.: Издательский дом «Вильямс», 2005, 544 с.

О чем курс?

О чем курс?

Уж точно не о сетевом администрировании!

О чем курс?

- Стек протоколов TCP/IP

О чем курс?

- Стек протоколов TCP/IP
- Сокеты – сетевой API

О чем курс?

- Стек протоколов TCP/IP
- Сокеты – сетевой API
- Архитектура клиент-сервер

О чем курс?

- Стек протоколов TCP/IP
- Сокеты – сетевой API
- Архитектура клиент-сервер
- Введение в распределенные системы

О чем курс?

- Стек протоколов TCP/IP
- Сокеты – сетевой API
- Архитектура клиент-сервер
- Введение в распределенные системы
- MPI – промышленный стандарт интерфейса для передачи сообщений

О чем курс?

- Стек протоколов TCP/IP
- Сокеты – сетевой API
- Архитектура клиент-сервер
- Введение в распределенные системы
- MPI – промышленный стандарт интерфейса для передачи сообщений
- Всякая околосетевая фигня с точки зрения программиста

О чем курс?

Поможет понимать такие шутки:

1: Я знаю замечательную шутку про UDP, но боюсь, что она до вас не дойдет.
2: Я знаю отличную шутку про TCP, но если она до вас не дойдет, то я повторю.
3. А кто знает отличную шутку про ARP? 4. А вы слышали шутку про ICMP? 5. Вам еще кто-то рассказывал шутку про STP? 6: Про MTU тоже есть кла

via bash(c)

Примеры

Все примеры рассчитаны, в первую очередь, на работу в среде POSIX

Проверялись на различных версиях ОС Linux, но должны работать на всех ОС, поддерживающих спецификацию POSIX (исключения будут оговариваться отдельно)

В основном используется язык Си, как наиболее приближенный к системному уровню

FAQ по работе примеров

- Q: А вот почему в моей любимой <здесь подставить название вашей любимой ~~говно~~ОС> не работает ваш пример?
  A: Смотрите документацию по API к своей ОС!
- Q: А вот почему в моём любимом языке <здесь подставить название вашего любимого языка> не получается реализовать <здесь подставить название фичи, которую не получается использовать>?
  A1: Смотрите документацию к своему языку!
  A2: А вы точно уверены, что выбранный вами язык вообще может работать на этом уровне?
  A3: А вы точно уверены, что вы знаете выбранный вами язык программирования?
  A4: И вообще с чего вы взяли, что я знаю этот язык программирования?

Главные ориентиры

- кроссплатформенная переносимость

Главные ориентиры

- кроссплатформенная переносимость
- открытые стандарты

Немного цитат

Дуг Макилрой, изобретатель каналов «pipes», сформулировал несколько постулатов,применимых для разработки ПО:

Немного цитат

Дуг Макилрой, изобретатель каналов «pipes», сформулировал несколько постулатов,применимых для разработки ПО:

- пишите программы, которые выполняют одну функцию и делают это хорошо;

Немного цитат

Дуг Макилрой, изобретатель каналов «pipes», сформулировал несколько постулатов,применимых для разработки ПО:

- пишите программы, которые выполняют одну функцию и делают это хорошо;
- пишите программы, которые будут работать вместе;

Немного цитат

Дуг Макилрой, изобретатель каналов «pipes», сформулировал несколько постулатов,применимых для разработки ПО:

- пишите программы, которые выполняют одну функцию и делают это хорошо;
- пишите программы, которые будут работать вместе;
- пишите программы, поддерживающие текстовые потоки, поскольку они являются универсальным интерфейсом.

"Философия"UNIX

это **не** философия, а общие рекомендации по проектированию ПО, накопленные сообществом программистов на опыте десятилетий разработок программ, которые взаимодействуют друг с другом.

1. Правило модульности

Следует писать простые части, связанные ясными интерфейсами.

1. Правило модульности

Следует писать простые части, связанные ясными интерфейсами.

Единственным способом создания сложной программы, не обреченной заранее на провал, является сдерживание ее глобальной сложности.

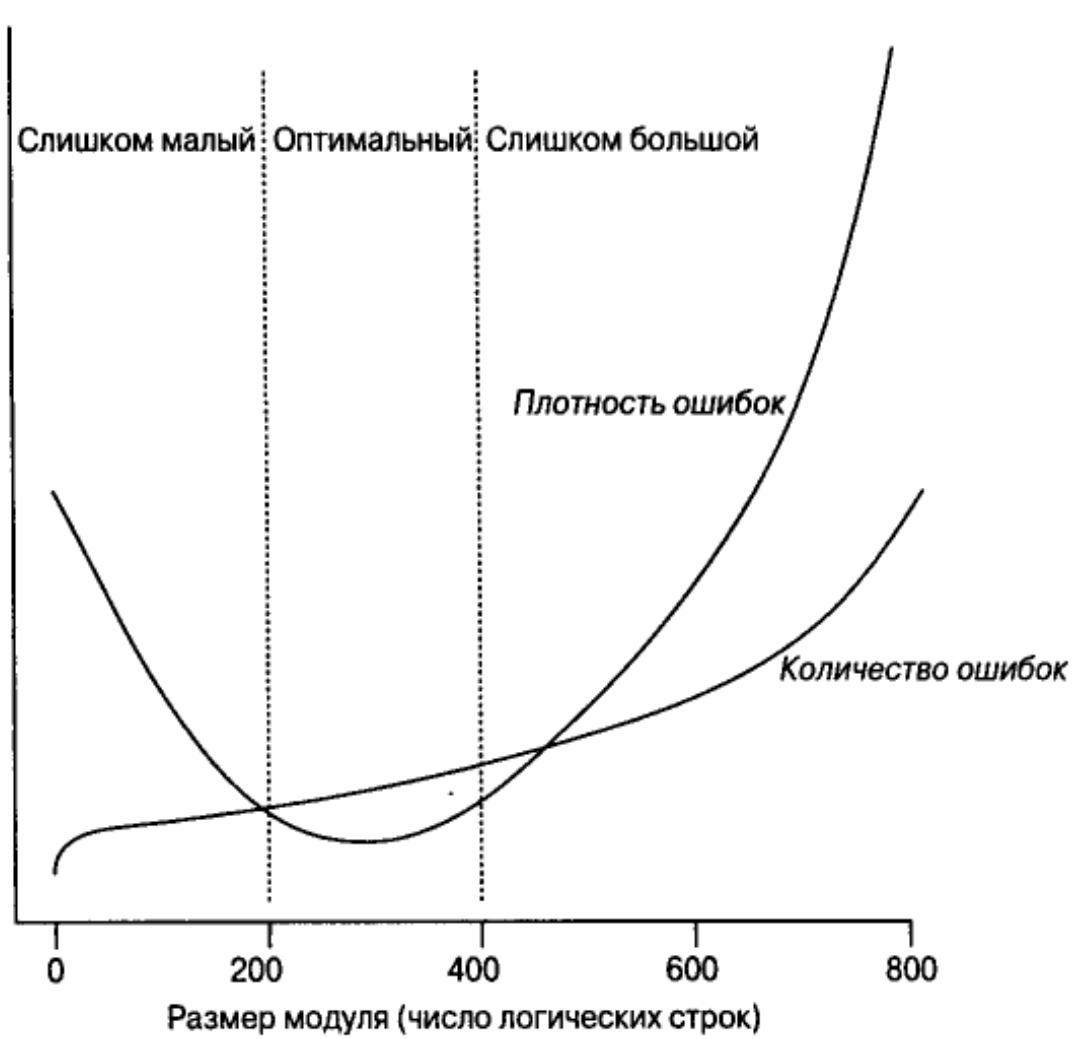
1. Правило модульности

Следует писать простые части, связанные ясными интерфейсами.

Единственным способом создания сложной программы, не обреченной заранее на провал, является сдерживание ее глобальной сложности.

Т.е. построение программы из простых частей, соединенных четко определенными интерфейсами, так что большинство проблем являются локальными, и тогда можно рассчитывать на обновление одной из частей без разрушения целого.

Размер кода и ошибки



2. Правило ясности

Ясность – лучше чем мастерство.

2. Правило ясности

Ясность – лучше чем мастерство.

Последующее обслуживание программы – важная и дорогостоящая часть жизненного цикла программы.

2. Правило ясности

Ясность – лучше чем мастерство.

Последующее обслуживание программы – важная и дорогостоящая часть жизненного цикла программы.

Писать программы необходимо так, как если бы вы знали, что последующей поддержкой будет заниматься неуравновешенный псих с топором, знающий ваш домашний адрес!

3. Правило композиции

Следует разрабатывать программы, которые будут взаимодействовать с другими программами.

3. Правило композиции

Следует разрабатывать программы, которые будут взаимодействовать с другими программами.

Если разрабатываемые программы не способны взаимодействовать друг с другом, то очень трудно избежать создания сложных монолитных программ.

3. Правило композиции

Следует разрабатывать программы, которые будут взаимодействовать с другими программами.

Если разрабатываемые программы не способны взаимодействовать друг с другом, то очень трудно избежать создания сложных монолитных программ.

Методы взаимодействия могут быть сильными и слабыми – по возможности рекомендуется использовать слабые методы и текстовые форматы передачи данных.

4. Правило разделения

Следует отделять политику от механизма и интерфейсы от основных модулей (engine).

Примеры политики и механизма:
вид GUI и операции отрисовки, клиент (front-end) – сервер (back-end), сценарии и библиотеки и др.

4. Правило разделения

Следует отделять политику от механизма и интерфейсы от основных модулей (engine).

Примеры политики и механизма:
вид GUI и операции отрисовки, клиент (front-end) – сервер (back-end), сценарии и библиотеки и др.

При жесткой связи политики и механизма:

- политика становится негибкой и усложняется ее изменение;
- изменение политики имеет строгую тенденцию к дестабилизации механизмов.

5. Правило простоты

Необходимо проектировать простые программы и «добавлять сложность» только там, где это необходимо.

5. Правило простоты

Необходимо проектировать простые программы и «добавлять сложность» только там, где это необходимо.

Основные причины добавления сложности:

- человеческий фактор (часто – желание «выпендриться»);
- проектные требования, продиктованные текущей модой, маркетингом или «левой пяткой заказчика»;

6. Правило расчетливости

Пишите большие программы, только если после демонстрации становится ясно, что ничего другого не остается.

Под «большими программами» здесь понимаются программы с большим объемом кода и значительной внутренней сложностью.

7. Правило прозрачности

Для того, чтобы упростить проверку и отладку программы, ее конструкция должна быть обозримой.

7. Правило прозрачности

Для того, чтобы упростить проверку и отладку программы, ее конструкция должна быть обозримой.

Программа *прозрачна*, если при ее минимальном изучении можно понять, что она делает и как.

7. Правило прозрачности

Для того, чтобы упростить проверку и отладку программы, ее конструкция должна быть обозримой.

Программа *прозрачна*, если при ее минимальном изучении можно понять, что она делает и как.
Программа *воспринимаема*, когда она имеет средства для мониторинга и отображения внутреннего состояния.

7. Правило прозрачности

Для того, чтобы упростить проверку и отладку программы, ее конструкция должна быть обозримой.

Программа *прозрачна*, если при ее минимальном изучении можно понять, что она делает и как.
Программа *воспринимаема*, когда она имеет средства для мониторинга и отображения внутреннего состояния.

Необходимо использовать достаточно простые форматы входных и выходных данных.
Интерфейс должен быть приспособлен для использования в отладочных сценариях.

8. Правило устойчивости

Устойчивость – следствие прозрачности и простоты.

8. Правило устойчивости

Устойчивость – следствие прозрачности и простоты.

Программа является *устойчивой*, когда она выполняет свои функции в неожиданных условиях, которые выходят за рамки предположений разработчика, как и в нормальных условиях.

8. Правило устойчивости

Устойчивость – следствие прозрачности и простоты.

Программа является *устойчивой*, когда она выполняет свои функции в неожиданных условиях, которые выходят за рамки предположений разработчика, как и в нормальных условиях. Программа является *простой*, если происходящее в ней не представляется сложным для восприятия человеком.

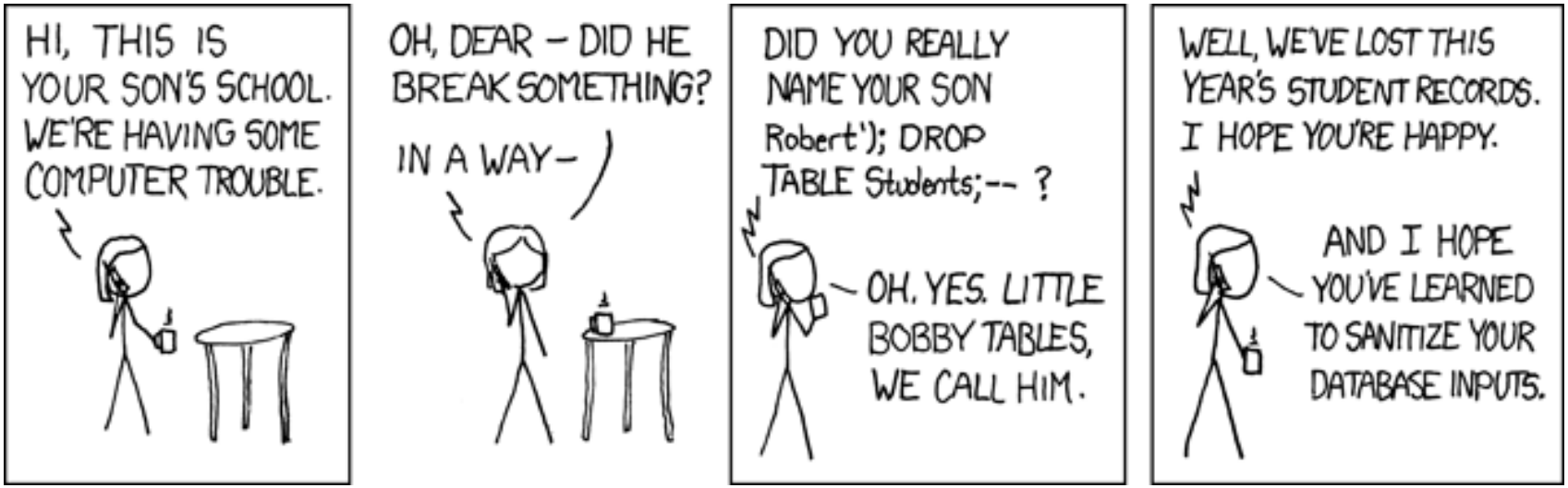
8. Правило устойчивости

Устойчивость – следствие прозрачности и простоты.

Программа является *устойчивой*, когда она выполняет свои функции в неожиданных условиях, которые выходят за рамки предположений разработчика, как и в нормальных условиях.
Программа является *простой*, если происходящее в ней не представляется сложным для восприятия человеком.

Один из способов организации – модульность(простые блоки, ясные интерфейсы)
Следует избегать частных случаев!

Пример неусточивого ПО



Пример «простой» программы

```
++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++.++++++++++++++++++
++++++++++++.+++++++..+++.------------------
------------------------------------------
--------------.++++++++++++++++++++++++++
++++++++++++++++++++++++.++++++++++++++++++
++++++.+++.------.--------.------------------
------------------------------------------
----.-----------------------.
```

9. Правило представления

Знания следует оставлять в данных, чтобы логика программы могла быть примитивной и устойчивой.

9. Правило представления

Знания следует оставлять в данных, чтобы логика программы могла быть примитивной и устойчивой.

Даже простую логику бывает сложно проверить, но даже сложные структуры данных являются довольно простыми для моделирования и анализа (например диаграмма 50 узлов дерева и блок-схема 50 строк кода)

9. Правило представления

Знания следует оставлять в данных, чтобы логика программы могла быть примитивной и устойчивой.

Даже простую логику бывает сложно проверить, но даже сложные структуры данных являются довольно простыми для моделирования и анализа (например диаграмма 50 узлов дерева и блок-схема 50 строк кода)

Если можно выбирать между усложнением структуры данных и усложнением кода, то лучше выбирать первое.
Примеры: ascii, генератор html-таблицы.

10. Правило наименьшего удивления

При проектировании интерфейсов всегда следует использовать наименее неожиданные элементы.

10. Правило наименьшего удивления

При проектировании интерфейсов всегда следует использовать наименее неожиданные элементы.

Необходимо учитывать характер предполагаемой аудитории и традиции платформы.

10. Правило наименьшего удивления

При проектировании интерфейсов всегда следует использовать наименее неожиданные элементы.

Необходимо учитывать характер предполагаемой аудитории и традиции платформы.

Оборотная сторона: следует избегать создания внешне похожих вещей, слегка отличающихся в действительности, поскольку *кажущаяся привычность порождает ложные ожидания*.

11. Правило тишины

Если программе нечего сказать, то пусть лучше молчит.

11. Правило тишины

Если программе нечего сказать, то пусть лучше молчит.

Внимание и сосредоточенность пользователя – ценный и ограниченный ресурс, который требуется только в случае необходимости.

11. Правило тишины

Если программе нечего сказать, то пусть лучше молчит.

Внимание и сосредоточенность пользователя – ценный и ограниченный ресурс, который требуется только в случае необходимости.

Важная информация не должна смешиваться с подробными сведениями о работе программы.

12. Правило восстановления

Когда программа завершается аварийно, это должно происходить явно (шумно) и по возможности быстро.

12. Правило восстановления

Когда программа завершается аварийно, это должно происходить явно (шумно) и по возможности быстро.

Если программа не способна справиться с ошибкой, то необходимо завершить ее работу так, чтобы максимально упростить диагностику.

12. Правило восстановления

Когда программа завершается аварийно, это должно происходить явно (шумно) и по возможности быстро.

Если программа не способна справиться с ошибкой, то необходимо завершить ее работу так, чтобы максимально упростить диагностику.

Для сетевых служб следует следовать рекомендации Постела: «*Будьте либеральны к тому, что принимаете, и консервативны к тому, что отправляете*»

13. Правило экономии

Время программиста дорого – поэтому задача экономии его времени более приоритетна, по сравнению с экономией машинного времени.

13. Правило экономии

Время программиста дорого – поэтому задача экономии его времени более приоритетна, по сравнению с экономией машинного времени.

Компьютер железный – ему не скучно (с) программистская мудрость

13. Правило экономии

Время программиста дорого – поэтому задача экономии его времени более приоритетна, по сравнению с экономией машинного времени.

Компьютер железный – ему не скучно (с) программистская мудрость

Использование высокоуровневых языков и «обучение» машины выполнять больше низкоуровневой работы по программированию, что приводит к правилу 14.

14. Правило генерации

Избегайте кодирования вручную; если есть возможность – пишите программы для создания программ.

14. Правило генерации

Избегайте кодирования вручную; если есть возможность – пишите программы для создания программ.

Использование генераторов кода оправданно, когда они могут повысить уровень абстракции, т.е. когда язык спецификации для генератора проще, чем сгенерированный код, и код впоследствии не потребует ручной доработки.

14. Правило генерации

Избегайте кодирования вручную; если есть возможность – пишите программы для создания программ.

Использование генераторов кода оправданно, когда они могут повысить уровень абстракции, т.е. когда язык спецификации для генератора проще, чем сгенерированный код, и код впоследствии не потребует ручной доработки.

Примеры: грамматические и лексические анализаторы, генераторы make-файлов, построители GUI-интерфейсов.

15. Правило оптимизации

Сначала – опытный образец, потом – оптимизирование.

Добейтесь стабильной работы, только потом оптимизируйте.

15. Правило оптимизации

Сначала – опытный образец, потом – оптимизирование.

Добейтесь стабильной работы, только потом оптимизируйте.

Керниган и Плоджер:

90% актуальной и реальной функциональности лучше, чем 100% функциональности перспективной и сомнительной

15. Правило оптимизации

Сначала – опытный образец, потом – оптимизирование.

Добейтесь стабильной работы, только потом оптимизируйте.

Керниган и Плоджер:

90% актуальной и реальной функциональности лучше, чем 100% функциональности перспективной и сомнительной

Кнут:

преждевременная оптимизация – корень всех зол

15. Правило оптимизации

Сначала – опытный образец, потом – оптимизирование.

Добейтесь стабильной работы, только потом оптимизируйте.

Керниган и Плоджер:

90% актуальной и реальной функциональности лучше, чем 100% функциональности перспективной и сомнительной

Кнут:

преждевременная оптимизация – корень всех зол

Кент Бек (экстремальное программирование):

заставьте программу работать, заставьте работать ее верно, а затем сделайте ее быстрой

16. Правило разнообразия

Не следует доверять утверждениям о «единственно правильном пути».

16. Правило разнообразия

Не следует доверять утверждениям о «единственно правильном пути».

Никто не обладает умом, достаточным для оптимизации всего или для предвидения всех возможных вариантов использования создаваемой программы.

17. Правило расширяемости

Разрабатывайте для будущего. Оно наступит быстрее, чем вы думаете.

17. Правило расширяемости

Разрабатывайте для будущего. Оно наступит быстрее, чем вы думаете.

При проектировании протоколов или форматов файлов следует делать их самоописательными, для того, чтобы их можно было расширить.

17. Правило расширяемости

Разрабатывайте для будущего. Оно наступит быстрее, чем вы думаете.

При проектировании протоколов или форматов файлов следует делать их самоописательными, для того, чтобы их можно было расширить.

Всегда, следует либо включать номер версии, либо составлять формат из самодостаточных, самоописательных команд так, чтобы можно было легко добавить новые директивы, а старые удалить, «не сбивая с толку» код чтения формата.

Все правила сразу

K.I.S.S.

Все правила сразу

K.I.S.S.

Keep It Simple, Stupid!

Не повторяйтесь!

Правило

SPOT

Single Point of Truth

Не повторяйтесь!

Правило

SPOT

Single Point of Truth

Внутри системы каждый блок знаний должен иметь единственное, недвусмысленное и надежное представление.

Дублирование данных?

Если дублирование данных существует из-за необходимости иметь два различных представления в двух различных местах, то возможно ли написать функцию, средство или генератор кода для создания одного представления из другого или обоих из общего источника?

Дублирование данных?

Если документация дублирует данные из кода, то можно ли создать фрагменты документации из кода или наоборот, или и то, и другое из общего представления более высокого уровня?

Дублирование данных?

Если файлы заголовков и объявления интерфейсов дублируют сведения в реализации кода, то существует ли способ создания файлов заголовков и объявлений интерфейсов из данного кода?

SPOT для структур данных

«нет лишнего — нет путаницы»

Предпочтительна структура данных, состояния которой имеют однозначное соответствие с состояниями реальной системы, которая будет моделироваться.

Спасибо за внимание!
Вопросы?