***Содержание***

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

**ТРПП.317.13.КР**

Выполнил

Проверил

**Тема: «Объектно-ориентированное программирование»**

Лит.

Листов

*32*

Введение 3

Основные принципы:

1. Инкапсуляция 4

2. Наследование 9

3. Полиморфизм 16

Совмещенные код и данные 25

Объектно-ориентированная методология(ООМ) 26

Программа «Решение квадратного уравнения» 28

Заключение 31

Список литературы 32

***Что такое хорошая программа.***

Вопрос оценки качества программы (и квалификации написавшего ее разработчика) является весьма важным и затрагивается во всех методических исследованиях на тему программирования. Относительно формулировки Э.Йодана — *"самое важное свойство программы заключается в том, что она работает"* — в целом особых расхождений во мнениях нет. Он же приводит классический пример дискуссии о достоинствах программы:

* *Программист А: "Моя программа в десять раз быстрее вашей, и занимает в три раза меньше места памяти!"*
* *Программист Б: "Да, но ваша программа не работает, а моя — работает!"*

Казалось бы, данное определение является слишком очевидным, чтобы его приводить. Однако в жизни приходится очень часто сталкиваться с ситуацией, когда пытаются сравнивать несравнимые вещи. Например, когда противопоставляют то, что "можно сделать", с тем, что "уже давно сделано".

Изм.

Лист

№ докум.

Подпись

Дата

Лист

8

ТРПП.317.18.КР

***Соответствие спецификациям.***

Работоспособность программы нуждается в уточнении — она должна отвечать исходному техническому заданию (Т3), исходным спецификациям. И тут нужно иметь в виду, что ТЗ может меняться в процессе разработки программы. Данная проблема особенно актуальна для России и тем более — для внутрифирменных разработок.

Сошлюсь на мнение одного программиста, который занимался "внутрифирменными" разработками сначала в одном крупном российском банке, а последние три года в американской корпорации (в США). Ключевое отличие состоит в более тщательном (у американцев), просто педантичном

отношении к документальному оформлению всех этапов разработки. Причем в утвержденный документ изменения вносятся только в исключительных случаях.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

9

ТРПП.317.18.КР

Обилие бумаг просто поражает и выводит из себя новичков из России. Однако именно эта рутинная работа обеспечивает независимость процесса реализации проекта от поведения отдельных его исполнителей. Принятие решения о начале какого-либо проекта занимает в США гораздо больше времени, чем у нас, однако если решение принято, в 80-90% случаев оно доводится до конца. (Три года назад в России этот показатель для внутрифирменных работ не превышал 30-40%.)

Одна из российских проблем — весьма произвольное изменение исходного технического задания. По мнению многих экспертов, более высокий процент успешных проектов с использованием "аутсорсинга" определяется в первую очередь обязательным наличием проработанного задания и вообще формализацией взаимоотношений между заказчиком и исполнителем. Во многих случаях внутрикорпоративных работ этапы технического задания и технического проекта носят декларативный характер.

***Сроки исполнения***

Следующим ключевым качеством "хорошей" программы является ее выполнение в заданные сроки. Этот вопрос был важен всегда, но сейчас, в условиях динамичного развития нашей жизни и бизнеса, роль выполнения временных графиков резко возрастает.

Одним из главный вопросов, который уже более 30 лет обсуждается исследователями организации работы программистов, заключается в том, как оценить реальную трудоемкость проекта (соответственно определить сроки его выполнения и нужные ресурсы) и обеспечить управляемость процесса его реализации.

Если проанализировать процесс развития средств разработки за все эти годы, то можно сделать категорический вывод, что первоочередной задачей являлось повышение производительности труда программиста, в том числе за счет снижения эффективности создаваемого кода (снижение эффективности кода за последние десятилетия — при одинаковой производительности процессоров и на задачах одной категории сложности — представляется мне почти очевидным фактом).

Изм.

Лист

№ докум.

Подпись

Дата

Лист

10

ТРПП.317.18.КР

Более того, модернизация инструментария в первую очередь преследовала цель повышения производительности не столько отдельного человека, сколько всего коллектива разработчиков. В подтверждение этому можно привести немало примеров, когда новая версия инструмента (например, за счет использования унифицированных компонентов, исключения "хитрых" трюков и пр.) даже снижает эффективность работы индивидуума, но при этом усиливает устойчивость процесса разработки. Именно таким образом "наука-искусство" программирования превращалось в технологию. Например, чтобы компанию из трех (плохо управляемых) суперпрограммистов можно было заменить командой из десяти средних, но взаимозаменяемых и управляемых разработчиков.

**"Если проект начат, то он должен быть закончен".**

Не будем останавливаться на случае, когда проект просто досрочно закрывается (отсутствие денег повлекло за собой понимание того, что проект не нужен и пр.) — "незавершёнка" является хронической болезнью нашей экономики. Остановимся лишь на вопросе коренного изменения структуры проекта, в ущерб срокам его реализации.

Рассмотрим, например, такую ситуацию. Вы начали проект со сроком реализации в один год. Через полгода обнаружили, что применив, например, совсем другие алгоритмы (или средства разработки), можно выполнить проект за 9 месяцев и повысить эффективность результата на 30%. Стоит ли прекращать выполнение старого варианта и начинать новый? Конечно, в

такой ситуации необходимо учитывать многие другие "граничные условия", но в этой постановке задачи почти очевидно, что затраты на реализацию второго варианта (с учетом уже потраченных усилий) будут на 25% выше. К тому же не следует забывать о возможных потерях от упущенной выгоды за три месяца задержки.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

11

ТРПП.317.18.КР

Кроме того, следует иметь в виду другой аспект: на этом этапе работы вероятность успешного завершения в срок первого варианта оценивается в 80%, а второго — вдвое меньше. Поэтому в общем случае оптимальным решением является завершение начатого варианта, а уже потом изучения вопроса о целесообразности его модернизации.

Мне хотелось бы обратить внимание, что хотя я постоянно говорю "проект", на самом деле все выше сказанное непосредственно относится и к разработке даже относительно небольших программ. И наоборот, многое из того, что говорится применительно к программированию, касается и реализации крупных проектов. Одним из вариантов разрешения противоречия ограниченности сроков и необходимости получения завершенного проекта является пошаговое его выполнение с получением конкретного практического результата на каждом этапе, то есть реализация проекта в виде поэтапного наращивания числа реализуемых функций.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

12

ТРПП.317.18.КР

***Простота в сопровождении и модернизации.***

Любое ранжирование носит субъективный характер, и самое главное здесь то, что выбор ценностей определяется с точки зрения заданной целевой функции. Я поставила свойство простоты программы на третью позицию, имея в виду такую вводную: есть конкретная проблема — ее нужно решить.

Но если подходить со стратегической точки зрения и рассматривать решение конкретной проблемы лишь как некоторый жизненный этап, то на первое место я бы поставила именно данную характеристику программы.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

13

ТРПП.317.18.КР

Это даже важнее соответствия спецификациям, поскольку спецификации могут меняться (причем еще в процессе данного этапа работы). Это важнее сроков, потому что жизненный цикл полезной программы гораздо продолжительнее времени, отведенного на создание одной ее версии.

Честно говоря, зафиксировав этот тезис, я нахожусь в некотором замешательстве. С одной стороны, мне он представляется очевидным и не требующим доказательства, с другой — слишком часто встречаешься с примерами программ, которые достоинством простоты и наглядности никак не обладают. Прежде всего, это нужно для:

* обеспечения эффективного поэтапного проектирования и разработки;
* повышения надежности программы посредством использования оптимальных схем полного сквозного тестирования;
* минимизации затрат на отладку: поиск и исправление ошибок;
* упрощения подготовки документации;
* обеспечения возможности повышения эффективности работы программ;
* функционального расширения программы в ближайшем и отдаленном будущем.

Обычно в качестве довода в пользу "простоты" говорится о необходимости поддержки коллективной работы — чтобы сопровождать и развивать программу могли бы другие разработчики. Но мне представляется, что каждый программист должен четко понять, что это нужно прежде всего

для обеспечения собственной эффективности (даже если он "программист-одиночка").

Изм.

Лист

№ докум.

Подпись

Дата

Лист

14

ТРПП.317.18.КР

**"Кто ясно мыслит, тот ясно излагает".**

К сожалению, среди разработчиков всегда были люди, которые создание запутанных программ (как они сами говорят, "изощренных") ставили себе в заслугу. Однако при ближайшем рассмотрении чаще всего оказывалось, что они запутывают не только окружающих, но и себя.

Понятие "простоты и наглядности" является, конечно, довольно расплывчатым, но я уверена, что искусство и технология программирования заключаются в значительной степени в достижении именно этой цели.

Надо сказать, что в аспекте данной проблемы работа современных программистов по сравнению с порой двадцатилетней давности в чем-то упростилась, а в чем-то усложнилась. Например, возможность использования длинных имен сильно упростила самодокументирование программ. Однако событийная модель приложения, с одной стороны, реализовала интуитивно понятную структуру программы, с другой — привела к появлению огромного числа небольших процедур, ориентироваться в которых не так-то просто. (Последнее усугубляется весьма слабым механизмом навигации по компонентам проекта, например в том же Visual Basic.)

Тут также следует вспомнить, что ранее программы имели лишь один уровень компонентов — процедуры (подпрограммы, функции). Однако еще в середине 80-х (например, в QuickBasic) появился еще один уровень — модули, которые стали состоять из набора процедур и внутренних переменных модуля. (Точнее, ранее понятие "модуля" было эквивалентно "процедуре" — в одном модуле могла находиться лишь одна процедура). Это, конечно, повышает гибкость разработки приложения, однако сразу возникает непростая задача оптимального распределения процедур по модулям.

Хотелось бы подчеркнуть, что понятие "простоты" в последнее время очень часто подменяется "примитивным" стилем программирования, который, к сожалению, присутствует и в учебной литературе.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

15

ТРПП.317.18.КР

***Эффективность.***

Изм.

Лист

№ докум.

Подпись

Дата

Лист

16

ТРПП.317.18.КР

Критерии эффективности должны быть обязательно заданы в исходных спецификациях проекта. То есть в общем случае эффективность должна соответствовать реальным требованиям, но не должна быть самоцелью. Обычно получение эффективного кода связывают с необходимостью повышения трудозатрат. И с учетом выбранных выше приоритетов стратегия разработки заключается в следующем: сначала создаем работоспособный вариант программы, а потом начинаем работать над повышением его эффективности.

Приходится довольно часто слышать, как программисты объясняли не очень высокую эффективность программы (и качество ее оформления) жесткими временными сроками. Но вот в чем парадокс: в большинстве случаев именно эффективный код и хорошая наглядность кода обеспечивают минимизацию временных затрат.

Разумеется, для сложных комплексов можно использовать специальные инструменты, но в действительности многие методы повышения эффективности кода хорошо известны на чисто теоретическом уровне. Просто нужно их знать и применять.

***Введение.***

Изм.

Лист

№ докум.

Подпись

Дата

Лист

5

ТРПП.317.18.КР

Когда компьютеры были большими, память у них была маленькой и   
быстродействие тоже. И поэтому гуру от этих ЭВМ ничего, кроме ассемблера (автокода) и не признавали. И на программистов смотрели свысока - писать, мол, эффективные программы не умеют, только зря машинное время расходуют.

И действительно, как было не признать! Пока программа небольшая и в память помещается — ладно, живите и радуйтесь! А как больше стала? Тут-то и придется признать свое бессилие.

Вряд ли кто-нибудь из нынешнего поколения слышал про "самоубийство программ". Нет, это не стирание EXE-файла во время его работы. Это когда программа во время своей работы использует память, занятую своими командами (которые уже отработали и больше не потребуются) под свои данные. Вряд ли кому-нибудь придется этим сейчас   
заниматься. А вот когда у Вас всего 18 Кбайт, то и таким не пренебрегали.

Потом компьютеры стали больше. И в размерах и память с быстродействием тоже росли.

Например, ЕС-1022. Памяти там было уже 512 Кбайт, 100 из них занимала ОС. Остальные 400 делились между 3-4 одновременно работавшими программами разных пользователей. Так что на тех, кто заказывал больше 200 Кбайт, на ВЦ смотрели довольно косо — за наш счет ведь!

Эффективность тогда была всегда на первом плане. Не как-нибудь написать, а наиболее эффективным способом, не любой алгоритм использовать, а лучший для этого случая. На этом прочно стояли.

В данной курсовой работе я бы хотела сделать упор именно на эффективность программ.

*"****По координатам точки и вершин треугольника определить, принадлежит ли точка этому треугольнику".***

Изм.

Лист

№ докум.

Подпись

Дата

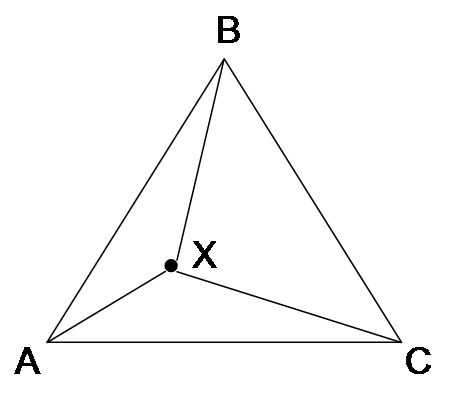
Лист

17

ТРПП.317.18.КР

  Начнем с того, каким образом это вообще можно сделать. Есть как минимум два варианта решения этой задачи.

***Вариант первый.***



Дан треугольник ABC и искомая точка X. Чтобы определить принадлежит ли точка треугольнику нужно проделать следующие действия:

1. Найти площадь треугольника ABC.
2. Найти площади треугольников ABX, BCX и ACX.
3. Сравнить. Если площадь треугольника ABC равна сумме площадей ABX, BCX и ACX, значит точка принадлежит треугольнику, иначе - нет.

***Второй вариант.***

Изм.

Лист

№ докум.

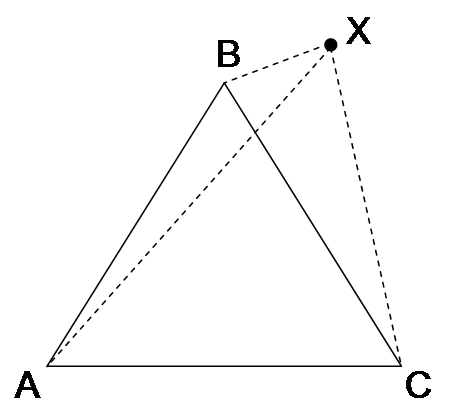
Подпись

Дата

Лист

18

ТРПП.317.18.КР



Ситуация такая же: ABC - треугольник, X - точка.

1. Находим площадь треугольника ABC.
2. Находим расстояние от точки X до каждой из вершин треугольника.
3. Выбираем наименьшее из них. (В данном случае BX)
4. Находим площадь треугольника с новой точкой вместо ближайшей. (Здесь XC)
5. Сравниваем. Если площадь первого треугольника больше площади второго, значит, точка не принадлежит треугольнику, иначе - принадлежит.

По сложности реализации оба варианта примерно одинаковы, но дело в том, что во втором варианте есть небольшой процент неточности - когда точка лежит очень близко к грани треугольника, но не принадлежит ему.

Поэтому подробно рассмотрим только первый вариант.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

19

ТРПП.317.18.КР

Выбор пути решения ясен. В любом случае в обоих случаях ответ находится через площадь. А как найти площадь по координатам точек? Ответ есть. Существует такая формула Герона.   
  
 ***Квадратный корень из p(p-x)(p-y)(p-z)***

Где p - полупериметр, x,y и z - длины сторон. Длину сторон же находятся по формуле длины вектора.

Первые строчки кода выглядят так:

*program Treugolnik;   
var q,w,p,ax,bx,cx,dx,ay,by,cy,dy:real;   
   per1,per2,per3,per4:real;*

  Переменные *q,w* промежуточные. В них будут храниться временные значения координат. Переменные *ax,bx...cy,dy* - координаты вершин треугольника и искомой точки. Переменные *per1...per4* и *p* будут служить для хранения площадей треугольников.

Сначала напишем процедуру получения координат с клавиатуры. Она выглядит так:

Изм.

Лист

№ докум.

Подпись

Дата

Лист

20

ТРПП.317.18.КР

*procedure Zapoln;   
begin   
write('Koordinaty A ');   
readln(ax, ay);   
write('Koordinaty B ');   
readln(bx, by);   
write('Koordinaty C ');   
readln(cx, cy);   
write('Koordinaty D ');   
readln(dx, dy);   
end;*

Координаты точки будут вводиться через пробел, при нажатии на Enter переход к следующей точке. Итак, мы получили координаты точек треугольников.

Теперь получим их площади.

*procedure Ploshad(n:integer);   
  var d1,d2,d3,per:real;   
begin   
  d1:=sqrt((ax-bx)\*(ax-bx)+(ay-by)\*(ay-by));   
  d2:=sqrt((cx-bx)\*(cx-bx)+(cy-by)\*(cy-by));   
  d3:=sqrt((ax-cx)\*(ax-cx)+(ay-cy)\*(ay-cy));   
  per:=(d1+d2+d3)/2;   
  per:=sqrt(per\*(per-d1)\*(per-d2)\*(per-d3));*

*if n=1 then per1:=per else   
  if n=2 then per2:=per else   
  if n=3 then per3:=per else   
  if n=4 then per4:=per;   
end;*

Изм.

Лист

№ докум.

Подпись

Дата

Лист

21

ТРПП.317.18.КР

*sqrt((ax-bx)\*(ax-bx)+(ay-by)\*(ay-by))* - формула нахождения длины вектора AB. *sqrt(per\*(per-d1)\*(per-d2)\*(per-d3))* - та самая формула Герона. Теперь при вызове этой процедуры мы будем указывать номер нужного треугольника. 1 - главный треугольник, 2-4 - треугольники образованные точкой.

Теперь нам нужно найти площади треугольников образованных точкой. Для этого будем временно подставлять вместо координат одной из вершин координаты точки D.

*procedure Zamena(x:integer);   
begin   
if x=1 then   
   begin   
   q:=cx;   
   w:=cy;   
   cx:=dx;   
   cy:=dy;   
   end else   
if x=2 then   
   begin   
   q:=ax;   
   w:=ay;*

*ax:=dx;   
   ay:=dy;   
   end else   
if x=3 then   
   begin   
   q:=bx;   
   w:=by;   
   bx:=dx;   
   by:=dy;   
   end;   
end;*

Изм.

Лист

№ докум.

Подпись

Дата

Лист

22

ТРПП.317.18.КР

Этой процедуре мы передаем номер операции. Я условно приняла, что нахождение площади ABD - 1, BCD - 2 и ACD - 3. В переменные *q* и *w* мы вводим временные координаты точки, чтобы потом можно было их восстановить.

Далее идет собственно сама процедура сравнения.

*procedure Sravn;   
begin   
  if per1 = p then   
  Writeln('Da, tochka prenadlegit treugolniku') else   
  Writeln('Net, tochka ne prenadlegit treugolniku'); end;*

*per1* - это площадь главного треугольника, а *p* - сумма площадей промежуточных треугольников. Но если мы будем сравнивать по этим данным, то ответ будет всегда отрицательным, так как после всех операций где-то далеко после запятой они все-таки расходятся.

Так что нам еще понадобится функция округления до сотых.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

23

ТРПП.317.18.КР

*function Okrug(z:real):real;   
  var a:integer;   
begin   
  z:=z/10\*1000;*    // После этого в целой части останутся первые 3 цифры

числа.   
*a:=round(z);*     // Получаем целую часть, т.е. отбрасываем все ненужное.   
  *z:=a/100;*    / Делим на 100 и получаем 2 знака после запятой.   
  *Okrug:=z;   
end;*

Написание всех необходимых процедур и функций закончено, теперь можно переходить к исполняемой части программы.

*Begin*   
*Zapoln;* // Получаем значения с клавиатуры.   
*Ploshad(1);* // Теперь в per1 находится площадь главного треугольника.   
*Zamena(1);* // Вместо точки C подставляем D.   
*Ploshad(2);* // Теперь в per2 находится площадь треугольника ABD.   
*cx:=q;* // Возвращаем точке C   
*cy:=w;* // ее координаты.   
*p:=per2;* // Пока сумма равна площади ABD.   
*Zamena(2);* // Вместо точки A подставляем D.   
*Ploshad(3);* // Теперь в per3 находится площадь треугольника BCD.   
*p:=p+per3;* // Сумма равна площадь ABD + площадь BCD.   
*ax:=q; //* Возвращаем точке A   
*ay:=w; //* ее координаты.   
*Zamena(3);* // Вместо точки B подставляем D.

*Ploshad(4);* // Теперь в per4 находится площадь треугольника ACD.   
*p:=p+per4;* // Теперь p = ABD + BCD + ACD.   
*p:=Okrug(p);* // Округляем сумму до сотых.   
*per1:=Okrug(per1);* // Округляем площадь главного треугольника до сотых.   
*Sravn;* // Сравниваем и выводим результат.   
*readln;* // Ждем нажатия клавиши Enter.   
*end.*

Изм.

Лист

№ докум.

Подпись

Дата

Лист

24

ТРПП.317.18.КР

***Предисловие.***

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

ТРПП.317.18.КР

***О пользе изучения программирования.***

Еще 10-15 лет назад считалось, что каждый технический специалист должен уметь худо-бедно делать программы для ЭВМ для решения каких-то своих задач. В такой постановке вопроса есть свой резон, ведь программирование — это, прежде всего создание логического алгоритма для достижения желаемой цели (с помощью компьютера или без него). Кроме того, даже если вы будете передавать задачу профессиональному разработчику, то знание основ программирования поможет правильно формулировать задание и вообще понимать "смежника".

Однако итоги всеобщего обучения программированию той поры вряд ли можно назвать успешными. На практике получалось так, что тот, кто хотел практически делать программы, осваивал эту технологию самостоятельно. Абсолютное же большинство студентов забывали о программировании вскоре после сдачи зачета. Так было в 80-х годах, так происходит и сейчас (я имею в виду подготовку некомпьютерных специалистов).

Причин этому довольно много. В 80-е годы (и ранее) освоение программирования напоминало изучение теории плавания без практических занятий на воде. К этому нужно прибавить откровенно слабую методику преподавания, которое сводилось не к изучению программирования на примере некоторого языка, а к освоению синтаксиса языка без привязки к технологии построения алгоритмов и практической реализации программ.

Сейчас компьютеры стали существенно доступнее, но ситуация с изучением программирования практически не улучшилась. Одной из причин этого является принципиальное сомнение в целесообразности его освоения. Действительно, зачем учиться программированию, если можно воспользоваться многочисленными готовыми приложениями? Поэтому

давайте просто обучать "компьютерной грамоте".

Да, такая постановка вопроса имеет рациональное зерно, но все же представляется довольно ограниченной.

1. Программирование помогает лучше формулировать логику решения практически любой задачи (совсем не обязательно чисто вычислительной). Как говорили в старину, "арифметика мозги в порядок приводит".
2. Расширение функциональности готовых программ с неизбежностью требует их более тонкой настройки и адаптации к нуждам конкретного пользователя. Использование методов программирования резко расширяет возможности решения этой задачи.
3. Привлечение для этого профессиональных разработчиков часто просто бессмысленно: сделать самому — быстрее, чем идти в другое крыло здания для общения с коллегами.

За последние 10 лет инструменты разработки и технология их освоения претерпели радикальное изменение. Эти средства стали существенно более понятными на чисто интуитивном уровне. Кроме того, радикально изменилась схема практической работы. Упрощенно говоря, раньше нужно было сначала изучить теорию программирования, чтобы написать даже простенькую программку, то сейчас серьезное освоение программирования начинается обычно после написания полезного приложения.

Современные средства позволяют ощутить эффект от программирования буквально с первых шагов, предоставляя возможность поэтапного наращивания своих знаний и навыков. Но при этом необходимо помнить одну важную истину: для перехода к серьезной профессиональной разработке одного опыта недостаточно — в какой-то момент потребуется изучение теории.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

4

ТРПП.317.18.КР

***Используемая литература.***

Изм.

Лист

№ докум.

Подпись

Дата

Лист

27

ТРПП.317.18.КР

1. Информатика, Н.В.Макарова, Л.А.Матвеев, В.Л.Бройдо, Москва 2005.
2. Статья «Поговорим о программировании», А.Колесов.
3. Статья «Об эффективности программ», П.Дворкин.
4. Интернет.

***О критериях эффективности в программировании.***

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

ТРПП.317.18.КР

В 50—60-х годах одной из основных характеристик сложности электронного оборудования могло служить число используемых в нем ламп, а затем полупроводниковых вентилей (диодов и транзисторов). Минимизация числа активных компонентов цифровых устройств, как правило, означала в то время существенное упрощение ЭВМ, повышение надежности.

Математический аппарат, который позволял проектировщику «экономить мышление» и минимизировать количество используемых в конструируемом приборе вентилей формальными методами, был в этих условиях полезным и широко использовался.

После появления па рубеже 70-х годов интегральных схем средней, а затем и большой (БИС) интеграции старая характеристика сложности цифрового устройства по числу используемых в нем диодов и транзисторов потеряла всякий смысл. С другой стороны, если в 50—60-х годах никому и в голову не могло прийти оценивать конструктивную сложность цифрового устройства суммарным числом «ножек» на цоколях всех его ламп или общим числом выводов на диодах и транзисторах, то в 70-х годах оказалось, что общее число активных компонентов уже не имеет решающего значения, а в большинстве случаев более важными параметрами являются количество выводов и число кристаллов.

В это время уже трудно было бы отыскать разработчика цифровой аппаратуры, который из любви к классическим методам ранее освоенного им математического аппарата занимался бы минимизацией числа транзисторов в цифровой аппаратуре четвертого поколения.

Совершенно иная ситуация сложилась в технологии программирования. Критерии эффективности программ, сформировавшиеся в 50-х годах для ламповых ЭВМ, с трудом «вы­травлялись» из науки и

практики программирования середины 70-х годов и все еще сохраняют живучесть в 80-х. Э. Йодан, защищая в 1977 г. структурное программирование от критиков, обращавших внимание ни очевидное снижение эффективности хорошо структурированных программ из-за увеличения числа обращений к подпрограммам, вынужден был ссылаться на энергичное замечание В. Вульфа:

Изм.

Лист

№ докум.

Подпись

Дата

Лист

7

ТРПП.317.18.КР

***«Во имя эффективности в программировании вычислений было совершено больше прегрешений (причем не всегда ее удавалось достичь), чем по какой-либо другой причине, включая непроходимую глупость».***

***Заключение.***

***Как же научиться быстро и эффективно создавать сложные программы, и при этом не терять интереса к своей работе?***

Изм.

Лист

№ докум.

Подпись

Дата

Лист

25

ТРПП.317.18.КР

Жесткие рекомендации (что и как надо делать) тут не подойдут, потому что у каждого человека своя, индивидуальная система внутренних ценностей и мотиваций. Программирование невозможно свести к набору готовых универсальных приемов и рецептов на все случаи, которые достаточно заучить и механически внедрить.

Наивысшей эффективности можно добиться только одним путем: разработка программы должна стать процессом, приближенным к реальной ситуации и поэтому максимально гибким. Излишне формальные методы и приемы программирования позволяют накопить знания в своей профессиональной сфере, но они не помогут воплотить эти знания в практику и научиться синтезировать верные решения, так как основаны на шаблонах, редко приближенных к конкретным ситуациям.

Мастерство программирования неправильно сводить к набору готовых рецептов действий во всех мыслимых ситуациях. В учебниках содержатся достаточно общие советы и приводятся примеры, подходящие для условий, которые никогда не совпадут с конкретными условиями конкретного разработчика. Мастерство программирования представляет собой систему интеллектуальных и поведенческих навыков, которые, как объясняет современная психология, опираются на бессознательную компетентность.

Навыки программирования должны уйти в подсознание. Для этого необходим труд, постоянная практика, сознательные усилия, — прежде всего, по совершенствованию самого себя.

Иначе знания, почерпнутые из книг, останутся мертвой информацией. А чтобы этого не случилось, постарайтесь осознать самое главное правило программиста**:**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

26

ТРПП.317.18.КР

***исходя из собственной мотивации, искать творческие идеи по реализации поставленной задачи и применять их на практике*.**