

Технологии разработки программного обеспечения (ИСиТ)

ВВЕДЕНИЕ. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММНОЙ ИНЖЕНЕРИИ (SOFTWARE ENGINEERING)

Программная инженерия (промышленное программирование) ассоциируется с разработкой сложных программ коллективами разработчиков.

Проблемы становления и развития отрасли – высокая стоимость программного обеспечения, сложность его создания, необходимость управления и прогнозирования процессов разработки.

Цель программной инженерии – сокращение стоимости программ.

I. История. Программирование – стадии эволюции

50-е годы 20 века.

Программирование в машинном коде для решения, научно-технических задач (расчет по формулам).

Наличие достаточно четко сформулированного технического задания.

Отсутствие этапа проектирования.

Составление документации после завершения разработки.

Зарождение концепции модульного программирования.

ПРОЦЕССОР		Машинный код		Ассемблер	
Устройство управления	Регистры: Рег. A, Рег. B, Рег. C, Рег. D, Рег. E, Рег. F, Рег. G, Рег. H, Рег. I, Рег. J, Рег. K, Рег. L, Рег. M, Рег. N, Рег. O, Рег. P, Рег. Q, Рег. R, Рег. S, Рег. T, Рег. U, Рег. V, Рег. W, Рег. X, Рег. Y, Рег. Z, Рег. AA, Рег. AB, Рег. AC, Рег. AD, Рег. AE, Рег. AF, Рег. AG, Рег. AH, Рег. AI, Рег. AJ, Рег. AK, Рег. AL, Рег. AM, Рег. AN, Рег. AO, Рег. AP, Рег. AQ, Рег. AR, Рег. AS, Рег. AT, Рег. AU, Рег. AV, Рег. AW, Рег. AX, Рег. AY, Рег. AZ, Рег. BA, Рег. BB, Рег. BC, Рег. BD, Рег. BE, Рег. BF, Рег. BG, Рег. BH, Рег. BI, Рег. BJ, Рег. BK, Рег. BL, Рег. BM, Рег. BN, Рег. BO, Рег. BP, Рег. BQ, Рег. BR, Рег. BS, Рег. BT, Рег. BU, Рег. BV, Рег. BW, Рег. BX, Рег. BY, Рег. BZ, Рег. CA, Рег. CB, Рег. CC, Рег. CD, Рег. CE, Рег. CF, Рег. CG, Рег. CH, Рег. CI, Рег. CJ, Рег. CK, Рег. CL, Рег. CM, Рег. CN, Рег. CO, Рег. CP, Рег. CQ, Рег. CR, Рег. CS, Рег. CT, Рег. CU, Рег. CV, Рег. CW, Рег. CX, Рег. CY, Рег. CZ, Рег. DA, Рег. DB, Рег. DC, Рег. DD, Рег. DE, Рег. DF, Рег. DG, Рег. DH, Рег. DI, Рег. DJ, Рег. DK, Рег. DL, Рег. DM, Рег. DN, Рег. DO, Рег. DP, Рег. DQ, Рег. DR, Рег. DS, Рег. DT, Рег. DU, Рег. DV, Рег. DW, Рег. DX, Рег. DY, Рег. DZ, Рег. EA, Рег. EB, Рег. EC, Рег. ED, Рег. EE, Рег. EF, Рег. EG, Рег. EH, Рег. EI, Рег. EJ, Рег. EK, Рег. EL, Рег. EM, Рег. EN, Рег. EO, Рег. EP, Рег. EQ, Рег. ER, Рег. ES, Рег. ET, Рег. EU, Рег. EV, Рег. EW, Рег. EX, Рег. EY, Рег. EZ, Рег. FA, Рег. FB, Рег. FC, Рег. FD, Рег. FE, Рег. FF, Рег. FG, Рег. FH, Рег. FI, Рег. FJ, Рег. FK, Рег. FL, Рег. FM, Рег. FN, Рег. FO, Рег. FP, Рег. FQ, Рег. FR, Рег. FS, Рег. FT, Рег. FU, Рег. FV, Рег. FW, Рег. FX, Рег. FY, Рег. FZ, Рег. GA, Рег. GB, Рег. GC, Рег. GD, Рег. GE, Рег. GF, Рег. GG, Рег. GH, Рег. GI, Рег. GJ, Рег. GK, Рег. GL, Рег. GM, Рег. GN, Рег. GO, Рег. GP, Рег. GQ, Рег. GR, Рег. GS, Рег. GT, Рег. GU, Рег. GV, Рег. GW, Рег. GX, Рег. GY, Рег. GZ, Рег. HA, Рег. HB, Рег. HC, Рег. HD, Рег. HE, Рег. HF, Рег. HG, Рег. HH, Рег. HI, Рег. HJ, Рег. HK, Рег. HL, Рег. HM, Рег. HN, Рег. HO, Рег. HP, Рег. HQ, Рег. HR, Рег. HS, Рег. HT, Рег. HU, Рег. HV, Рег. HW, Рег. HX, Рег. HY, Рег. HZ, Рег. IA, Рег. IB, Рег. IC, Рег. ID, Рег. IE, Рег. IF, Рег. IG, Рег. IH, Рег. II, Рег. IJ, Рег. IK, Рег. IL, Рег. IM, Рег. IN, Рег. IO, Рег. IP, Рег. IQ, Рег. IR, Рег. IS, Рег. IT, Рег. IU, Рег. IV, Рег. IW, Рег. IX, Рег. IY, Рег. IZ, Рег. JA, Рег. JB, Рег. JC, Рег. JD, Рег. JE, Рег. JF, Рег. JG, Рег. JH, Рег. JI, Рег. JJ, Рег. JK, Рег. JL, Рег. JM, Рег. JN, Рег. JO, Рег. JP, Рег. JQ, Рег. JR, Рег. JS, Рег. JT, Рег. JU, Рег. JV, Рег. JW, Рег. JX, Рег. JY, Рег. JZ, Рег. KA, Рег. KB, Рег. KC, Рег. KD, Рег. KE, Рег. KF, Рег. KG, Рег. KH, Рег. KI, Рег. KJ, Рег. KK, Рег. KL, Рег. KM, Рег. KN, Рег. KO, Рег. KP, Рег. KQ, Рег. KR, Рег. KS, Рег. KT, Рег. KU, Рег. KV, Рег. KW, Рег. KX, Рег. KY, Рег. KZ, Рег. LA, Рег. LB, Рег. LC, Рег. LD, Рег. LE, Рег. LF, Рег. LG, Рег. LH, Рег. LI, Рег. LJ, Рег. LK, Рег. LL, Рег. LM, Рег. LN, Рег. LO, Рег. LP, Рег. LQ, Рег. LR, Рег. LS, Рег. LT, Рег. LU, Рег. LV, Рег. LW, Рег. LX, Рег. LY, Рег. LZ, Рег. MA, Рег. MB, Рег. MC, Рег. MD, Рег. ME, Рег. MF, Рег. MG, Рег. MH, Рег. MI, Рег. MJ, Рег. MK, Рег. ML, Рег. MM, Рег. MN, Рег. MO, Рег. MP, Рег. MQ, Рег. MR, Рег. MS, Рег. MT, Рег. MU, Рег. MV, Рег. MW, Рег. MX, Рег. MY, Рег. MZ, Рег. NA, Рег. NB, Рег. NC, Рег. ND, Рег. NE, Рег. NF, Рег. NG, Рег. NH, Рег. NI, Рег. NJ, Рег. NK, Рег. NL, Рег. NM, Рег. NN, Рег. NO, Рег. NP, Рег. NQ, Рег. NR, Рег. NS, Рег. NT, Рег. NU, Рег. NV, Рег. NW, Рег. NX, Рег. NY, Рег. NZ, Рег. OA, Рег. OB, Рег. OC, Рег. OD, Рег. OE, Рег. OF, Рег. OG, Рег. OH, Рег. OI, Рег. OJ, Рег. OK, Рег. OL, Рег. OM, Рег. ON, Рег. OO, Рег. OP, Рег. OQ, Рег. OR, Рег. OS, Рег. OT, Рег. OU, Рег. OV, Рег. OW, Рег. OX, Рег. OY, Рег. OZ, Рег. PA, Рег. PB, Рег. PC, Рег. PD, Рег. PE, Рег. PF, Рег. PG, Рег. PH, Рег. PI, Рег. PJ, Рег. PK, Рег. PL, Рег. PM, Рег. PN, Рег. PO, Рег. PP, Рег. PQ, Рег. PR, Рег. PS, Рег. PT, Рег. PU, Рег. PV, Рег. PW, Рег. PX, Рег. PY, Рег. PZ, Рег. QA, Рег. QB, Рег. QC, Рег. QD, Рег. QE, Рег. QF, Рег. QG, Рег. QH, Рег. QI, Рег. QJ, Рег. QK, Рег. QL, Рег. QM, Рег. QN, Рег. QO, Рег. QP, Рег. QQ, Рег. QR, Рег. QS, Рег. QT, Рег. QU, Рег. QV, Рег. QW, Рег. QX, Рег. QY, Рег. QZ, Рег. RA, Рег. RB, Рег. RC, Рег. RD, Рег. RE, Рег. RF, Рег. RG, Рег. RH, Рег. RI, Рег. RJ, Рег. RK, Рег. RL, Рег. RM, Рег. RN, Рег. RO, Рег. RP, Рег. RQ, Рег. RR, Рег. RS, Рег. RT, Рег. RU, Рег. RV, Рег. RW, Рег. RX, Рег. RY, Рег. RZ, Рег. SA, Рег. SB, Рег. SC, Рег. SD, Рег. SE, Рег. SF, Рег. SG, Рег. SH, Рег. SI, Рег. SJ, Рег. SK, Рег. SL, Рег. SM, Рег. SN, Рег. SO, Рег. SP, Рег. SQ, Рег. SR, Рег. SS, Рег. ST, Рег. SU, Рег. SV, Рег. SW, Рег. SX, Рег. SY, Рег. SZ, Рег. TA, Рег. TB, Рег. TC, Рег. TD, Рег. TE, Рег. TF, Рег. TG, Рег. TH, Рег. TI, Рег. TJ, Рег. TK, Рег. TL, Рег. TM, Рег. TN, Рег. TO, Рег. TP, Рег. TQ, Рег. TR, Рег. TS, Рег. TT, Рег. TU, Рег. TV, Рег. TW, Рег. TX, Рег. TY, Рег. TZ, Рег. UA, Рег. UB, Рег. UC, Рег. UD, Рег. UE, Рег. UF, Рег. UG, Рег. UH, Рег. UI, Рег. UJ, Рег. UK, Рег. UL, Рег. UM, Рег. UN, Рег. UO, Рег. UP, Рег. UQ, Рег. UR, Рег. US, Рег. UT, Рег. UY, Рег. UZ, Рег. VA, Рег. VB, Рег. VC, Рег. VD, Рег. VE, Рег. VF, Рег. VG, Рег. VH, Рег. VI, Рег. VJ, Рег. VK, Рег. VL, Рег. VM, Рег. VN, Рег. VO, Рег. VP, Рег. VQ, Рег. VR, Рег. VS, Рег. VT, Рег. VU, Рег. VV, Рег. VW, Рег. VX, Рег. VY, Рег. VZ, Рег. WA, Рег. WB, Рег. WC, Рег. WD, Рег. WE, Рег. WF, Рег. WG, Рег. WH, Рег. WI, Рег. WJ, Рег. WK, Рег. WL, Рег. WM, Рег. WN, Рег. WO, Рег. WP, Рег. WQ, Рег. WR, Рег. WS, Рег. WT, Рег. WY, Рег. WZ, Рег. XA, Рег. XB, Рег. XC, Рег. XD, Рег. XE, Рег. XF, Рег. XG, Рег. XH, Рег. XI, Рег. XJ, Рег. XK, Рег. XL, Рег. XM, Рег. XN, Рег. XO, Рег. XP, Рег. XQ, Рег. XR, Рег. XS, Рег. XT, Рег. XU, Рег. XV, Рег. XW, Рег. XX, Рег. XY, Рег. XZ, Рег. YA, Рег. YB, Рег. YC, Рег. YD, Рег. YE, Рег. YF, Рег. YG, Рег. YH, Рег. YI, Рег. YJ, Рег. YK, Рег. YL, Рег. YM, Рег. YN, Рег. YO, Рег. YP, Рег. YQ, Рег. YR, Рег. YS, Рег. YT, Рег. YU, Рег. YV, Рег. YW, Рег. YX, Рег. YY, Рег. YZ, Рег. ZA, Рег. ZB, Рег. ZC, Рег. ZD, Рег. ZE, Рег. ZF, Рег. ZG, Рег. ZH, Рег. ZI, Рег. ZJ, Рег. ZK, Рег. ZL, Рег. ZM, Рег. ZN, Рег. ZO, Рег. ZP, Рег. ZQ, Рег. ZR, Рег. ZS, Рег. ZT, Рег. ZU, Рег. ZV, Рег. ZW, Рег. ZX, Рег. ZY, Рег. ZZ	0E 1F BA0E00 B409 CD21 B8014C CD21 54 68 69 7320 7072 6F 67 7261 6D 206361 6E 6E 6F	PUSH CS POP DS MOV DX, 000E MOV AH, 09 INT 21 MOV AX, 4C01 TNT SP PUSH SP JNB 0033 JO 00B7 DB 6F DB 67 JB 007A DB 6D AND [BP+DI+61], AH DB 6E DB 6E DB 6F		

60-е годы.

Широкое использование языков программирования высокого уровня (Алгол 60, Фортран, Кобол и др.).

Возрастание сложности задач, решаемых с помощью компьютеров.

Использование методов коллективной работы

<p>Fortran Father of FORTRAN: JOHN BACKUS</p>	<p>Код высшего порядка</p> <pre>print("Hello, World!")</pre> <p>Ассемблированный код</p> <pre>section .text global _start _start: mov edi, len mov esi, msg mov ebx, 1 mov eax, 4 int 0x80 mov ebx, 1 int 0x80 section .data msg db "Hello, world!", 0xa len equ \$ - msg</pre> <p>Машинный код</p> <pre>98 1000 99 23 55 10 2 96 2 32 12 45 11</pre> <p>Шиминтел</p> <p>Ой! Ша понятной!</p>	
--	---	--

70-е годы.

Широкое распространение информационных систем и баз данных.

Развитие абстрактных типов данных.

Исследование проблем обеспечения надежности и мобильности программных средств.

Создание методики управления коллективной разработкой программ. Появление инструментальных средств поддержки программирования.

БД

содержит обширную информацию самого различного типа: текстовую, графическую, звуковую, мультимедийную.

Примеры:

1. в БД законов – тексты самих законов,
2. в БД эстрадной песни – тексты и ноты песен, биографию авторов, информация о поэтах, композиторах и исполнителях, звуковые и видеоклипы.

Абстрактный тип данных – это совокупность данных и операций над ними

Структура данных является частью реализации АТД

Перед реализацией АТД необходимо тщательно описать все операции, которые необходимо выполнять



80-е годы.

Широкое внедрение персональных компьютеров во все сферы человеческой деятельности.

Бурное развитие пользовательских интерфейсов и создание четкой концепции качества ПО.

Внедрение объектного подхода к разработке программных систем.

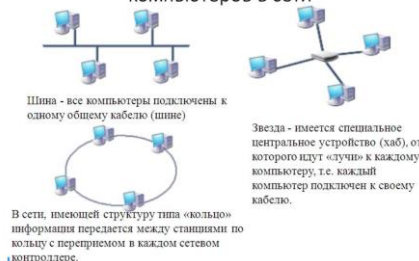
Развитие концепции компьютерных сетей.



Интерфейс включает в себя:

- способы взаимодействия с внутренней частью программы (операционной системой, платформой, сервером и т.д.);
- дизайн;
- доступные функции.

Топология сети – схема соединения компьютеров в сети



90-е годы.

В 1989 году реализован проект Всемирной паутины.

Актуальность проблемы защиты компьютерной информации и передаваемых по сети сообщений.

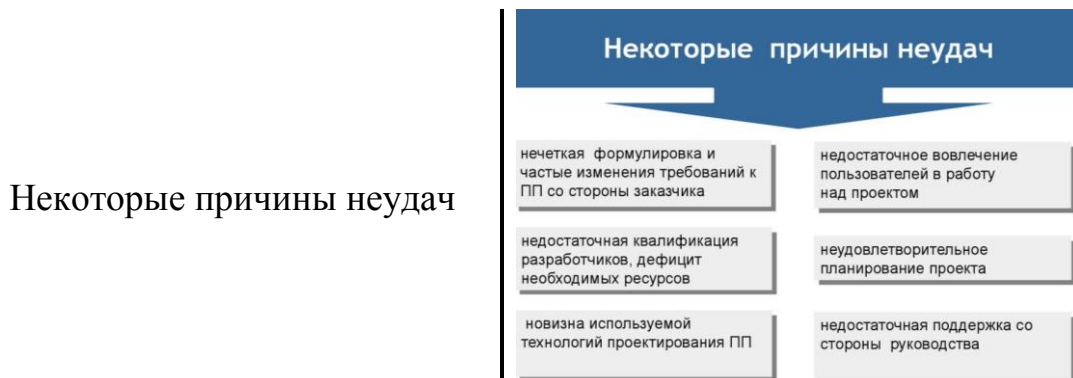
Развитие CASE-средств разработки программного обеспечения.

Глобальная сеть Internet



Разработка ПО по-прежнему остается непредсказуемой

Процент успешных проектов по созданию программного обеспечения достаточно низок:



II. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ:

Программа – это объект разработки, который не является осязаемым (нельзя пощупать, взвесить и т. п.), доступен пониманию ЭВМ, для которой написан.

Свойства хорошей программы

- Выполнение функциональных требований
- Соответствие нефункциональным требованиям
- Сопровождаемость (maintainability)
- Надежность (dependability)
- Эффективность (efficiency)
- Удобство использования (usability)

Программный продукт (ПП): программа, работающая без авторского присутствия. Программный продукт исполняется, тестируется, конфигурируется без присутствия автора и сопровождается документацией.

Программное обеспечение (ПО) – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ (ГОСТ 19781-90)

Программная инженерия (Software Engineering) ориентирована на разработку программного обеспечения прикладных и информационных систем разного назначения.

Определение из свода знаний по программной инженерии SWEBOOK:

- 1) **Программная инженерия** – это применение систематического, дисциплинированного и измеряемого подхода к разработке, эксплуатации и сопровождению программного обеспечения (ПО) с применением инженерных методов к разработке ПО.
- 2) **Программная инженерия** – учебная дисциплина, изучающая указанные выше подходы.

Отличие от других инженерий

Программная инженерия – это система методов, средств и дисциплин планирования, разработки, эксплуатации и сопровождения программного обеспечения, готового к внедрению.

Программа – не материальный объект: фазы производства и изготовления образца отсутствуют.

Стоимость программы зависит от стоимости и качества проектирования.

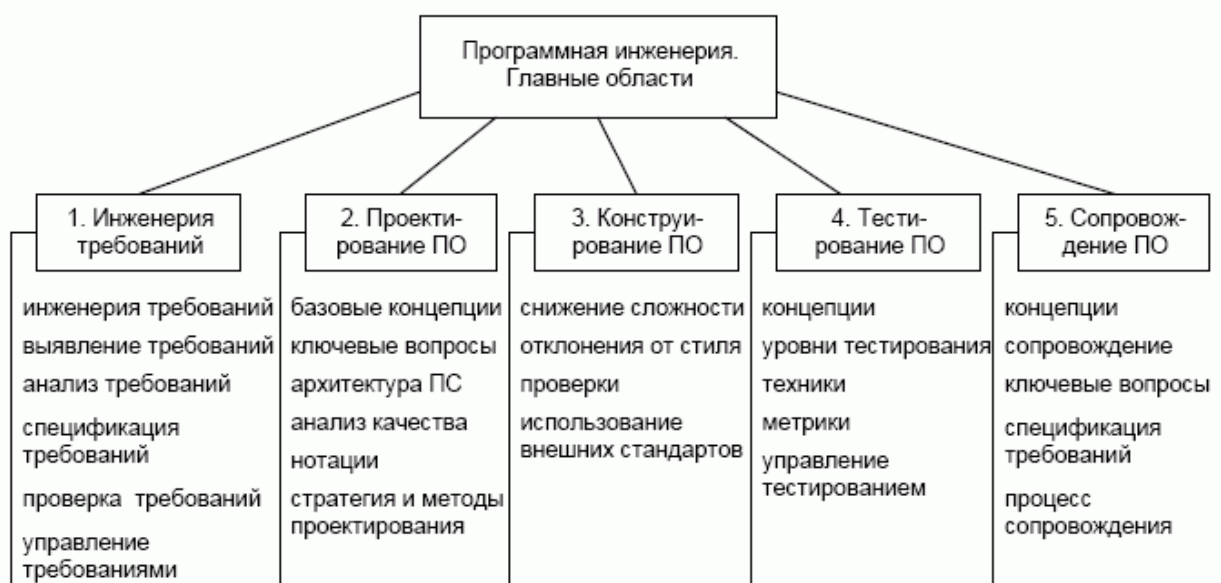
Нет объективных законов контроля проекта: тестирование – единственный способ проверки

Успех реализации проекта ПО обусловлен пятью взаимосвязанными аспектами:



Жизненный цикл ПО – непрерывный процесс с момента принятия решения о создании ПО до снятия его с эксплуатации.

Программная инженерия (Software Engineering)



Система программирования:

комплекс программных средств, предназначенных для автоматизации процесса разработки, отладки ПО и подготовки программного кода к выполнению



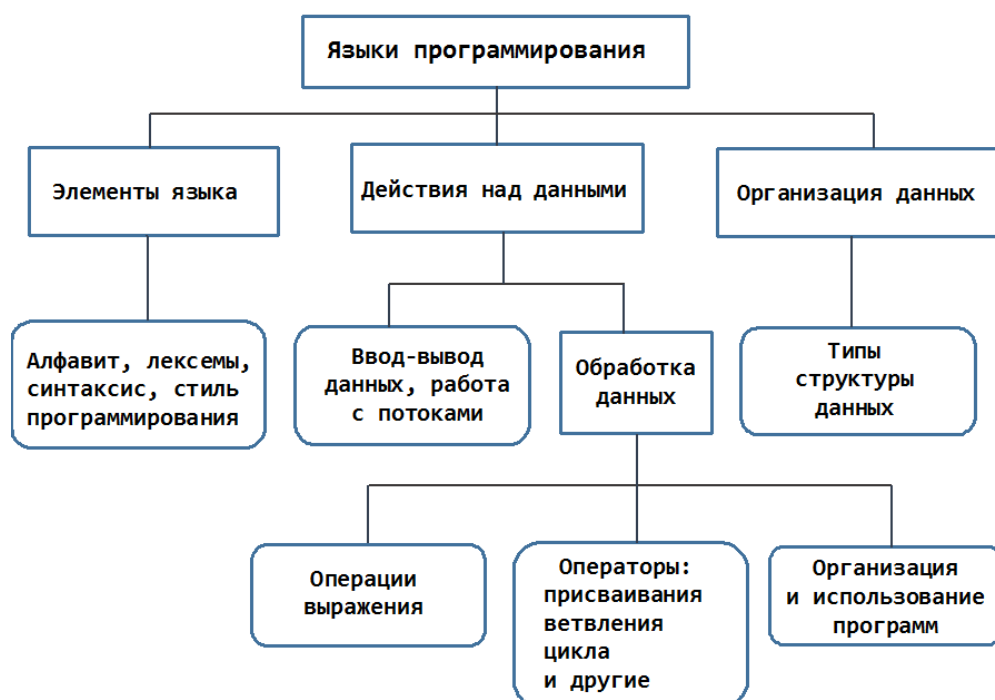
Интегрированная среда разработки:

набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

Примеры IDE (визуальные среды):

Eclipse, Microsoft Visual Studio, NetBeans, Qt Creator, ...

Структура языка программирования:



Программа – алгоритм, записанный на языке программирования.

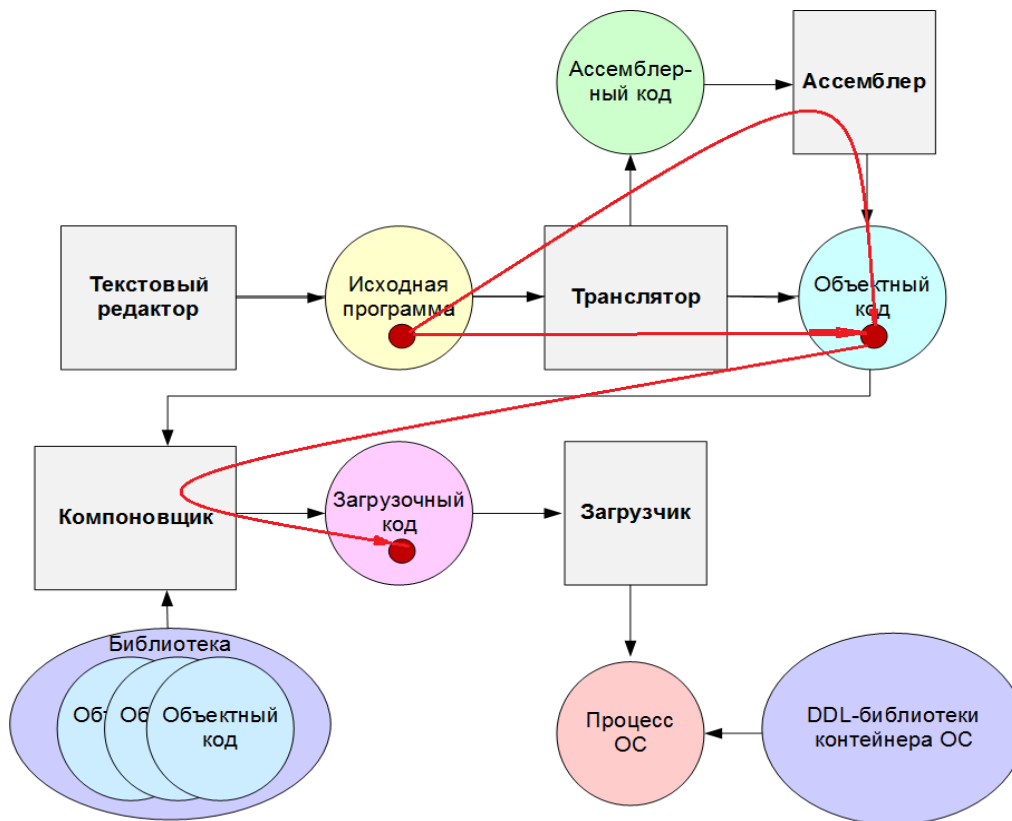
Текст программы (исходный код) – полное законченное и детальное описание алгоритма на языке программирования.

Объектный код: – результат работы транслятора. Один файл объектного кода – объектный модуль.

Объектный модуль – двоичный файл, который может быть объединён с другими объектными файлами при помощи редактора связей (компоновщика) для получения готового исполняемого модуля, либо библиотеки.

Загрузочный код – результат работы компоновщика.
Один файл загрузочного кода – загрузочный модуль.

От исходного кода к исполняемому модулю:


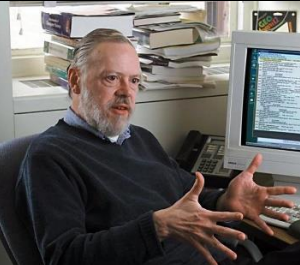


<p>Компилятор (транслятор) – программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке; результат – объектный модуль</p>	<p>Компоновщик (linker, редактор связей) – программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль</p>	<p>Загрузчик (loader) – программа, предназначенная для запуска процесса операционной системы на основе загрузочного модуля</p>
--	---	---

III. СИСТЕМЫ ПРОГРАММИРОВАНИЯ WINDOWS И UNIX

Unix («UNIX» – зарегистрированная торговая марка The Open Group) – семейство переносимых, многозадачных и многопользовательских операционных систем, которые основаны на идеях оригинального проекта AT&T Unix, разработанного в 1970-х годах в исследовательском центре Bell Labs Кеном Томпсоном, Деннисом Ритчи и другими.

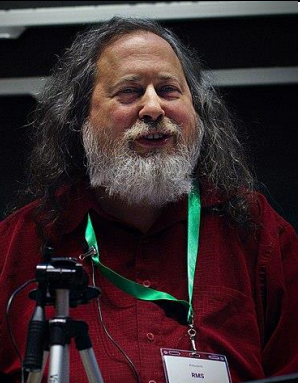
Дизайнеры языков программирования:

	Кеннет Лейн Томпсон – пионер компьютерной науки, известен своим вкладом в создание языка программирования C и операционной системы UNIX. Написал язык программирования B, предшественник языка C, участвует в создании языка программирования Go
	Деннис Ритчи – создатель языка программирования Си. Вместе с Кеном Томпсоном разработал Си для создания операционной системы UNIX. «У Ньютона есть фраза о стоящих на плечах гигантов. Мы все стоим на плечах Денниса», – Брайан Керниган.

Операционные системы семейства Unix характеризуются модульным дизайном, в котором каждая задача выполняется отдельной утилитой, взаимодействие осуществляется через единую файловую систему, а для работы с утилитами используется командная оболочка.

Unix является мультиплатформенной системой. Ядро системы разработано таким образом, что его легко можно приспособить практически под любой микропроцессор.

Linux (GNU/Linux) – семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта **GNU** (проект по разработке свободного программного обеспечения, запущенный известным программистом и сторонником СПО Ричардом Столлманом 27 сентября 1983 года в Массачусетском технологическом институте).

	Ричард Мэттью Столлман – основатель движения свободного программного обеспечения, проекта GNU , Фонда свободного программного обеспечения и Лиги за свободу программирования. Автор концепции «копилефта» (в противоположность традиционному подходу к авторскому праву, при котором ограничивается свобода копирования произведений, копилефт стремится использовать законы об авторском праве для расширения прав и свобод людей).
---	---

Linux-системы реализуются на модульных принципах, стандартах и соглашениях, заложенных в Unix. Подобные системы используют монолитное ядро для управления процессами, сетевыми функциями, периферией и доступом к файловой системе. Драйверы устройств либо интегрированы непосредственно в ядро, либо добавлены в виде модулей, загружаемых во время работы системы.

Linux доминирует на рынке интернет-серверов и смартфонов (операционная система Android имеет в основе Linux ядро). Linux полностью бесплатная система, в основном построенная на открытом программном обеспечении.

Принципы взаимодействия программ в Windows.

Интерфейс вызовов функций в Windows – доступ к системным ресурсам осуществляется через целый ряд системных функций. Совокупность таких функций называется прикладным программным интерфейсом или **API** (Application Programming Interface). Для взаимодействия с Windows приложение запрашивает функции API, с помощью которых реализуются все необходимые системные действия, такие как выделение памяти, вывод на экран, создание окон и т.п.

Функции API содержатся в библиотеках динамической загрузки (Dynamic Link Libraries, или DLL), которые загружаются в память только в тот момент, когда к ним происходит обращение, т.е. при выполнении программы.

Многозадачность в Windows

В Windows два типа многозадачности: основанный на процессах и основанный на потоках.

Процесс – это программа, которая выполняется. При многозадачности такого типа две или более программы могут выполняться параллельно.

Поток – это отдельная часть исполняемого кода. В многозадачности данного типа отдельные потоки внутри одного процесса также могут выполняться одновременно.

Стандарт POSIX

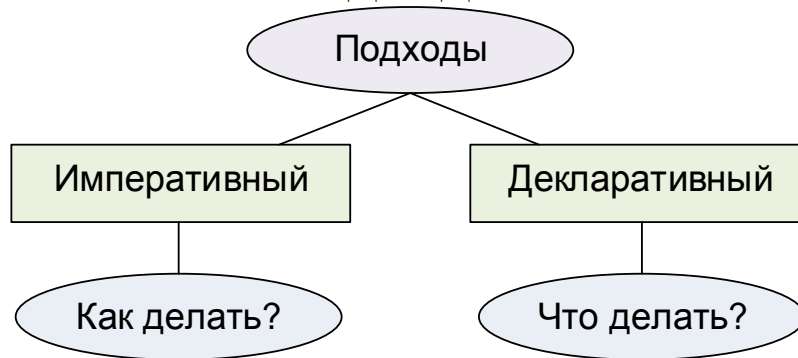
Стандарт POSIX – переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов.

Традиционно говорят о двух мирах, двух системах мировоззрения, присущих пользователям операционных систем Windows и UNIX.

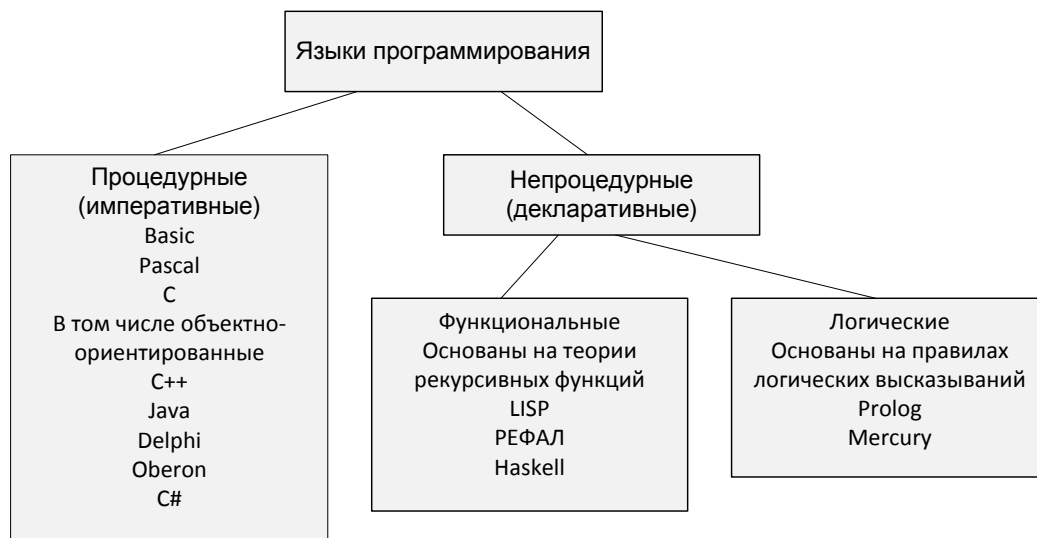
В 1985 году принят стандарт POSIX. Это стандарт на интерфейсы UNIX-подобных ОС.

Однако никто не запрещает системам, не являющимся клонами (потомками) UNIX, поддерживать стандарт POSIX.

IV. ОСНОВНЫЕ ПОДХОДЫ К ПРОГРАММИРОВАНИЮ:



ПАРАДИГМЫ (СТИЛИ) ПРОГРАММИРОВАНИЯ



Императивное программирование:

Программа = последовательность действий, дающих указания компьютеру о том, **как** получить решение.

Конструкции языка:

последовательность действий;

условная конструкция;

цикл.

Пример:

Функция с именем add, принимает на вход массив и возвращает сумму всех его элементов.

Псевдокод:

```
function add(array)
{
    let result = 0
    for (let i = 0; i < array.length; i++)
        result += array[i];
    return result;
}
```

Декларативное программирование:

Программа = описание действий, которые необходимо выполнить компилятору для получения результата.

Отвечает на вопрос **что** надо выполнить.

<p>Пример: Функция с именем add, принимает на вход массив и возвращает сумму всех его элементов.</p>	<p>Псевдокод: <pre>function add(array) { return array.reduce((prev, current) => prev + current, 0) }</pre> </p>
<p>Функциональное программирование: Программа = система определений и функций, описывающих что нужно вычислить, а как это сделать – решает транслятор; последовательность действий не прослеживается. Раздел дискретной математики. Основой функционального программирования является лямбда-исчисление</p>	
<p>Объектно-ориентированное программирование: Программа = несколько взаимодействующих объектов + функциональность (действия и данные распределяются между этими объектами).</p>	
<p>Распределённое (параллельное) программирование: Программа = совокупность описаний процессов, которые могут выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора).</p>	
<p>Логическое программирование: Программа = система определений вида «условие => новый факт». Программа – это описание фактов и правил вывода в некотором логическом исчислении. Результат получается системой путем логического вывода. Раздел математической логики.</p>	
<p>Распределённое (параллельное) программирование: Программа = совокупность описаний процессов, которые могут выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора).</p>	
<p>Визуальное программирование: Программа = способ создания программы для ЭВМ путём манипулирования графическими объектами вместо написания её текста. Визуальное программирование на уровне алгоритмов, а не программного кода. На основании составленной программистом «блок-схемы» в автоматическом режиме генерируется написанный на языках программирования (1GL, 2GL, 3GL),..</p>	
<p>Аспектно-ориентированное программирование: Программа = к уже существующему коду добавляется дополнительного поведение, так называемой сквозной функциональности.</p>	

Пример: функция возведения в квадрат.

Императивный C:	Функциональный Scheme:
<pre>int square(int x){ return x * x; }</pre>	<pre>(define square (lambda (x) (* x x)))</pre>
Конкатенативный Joy:	Конкатенативный Factor:
<pre>DEFINE square == dup * .</pre>	<pre>: square (x -- y) dup *;</pre>

Язык программирования строится в соответствии с базовой моделью вычислений и парадигмой программирования.

Парадигма программирования – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию).

Языки классифицируют по важнейшим признакам:

- **эволюционным** – поколения языков (1GL, 2GL, 3GL, 4GL, 5GL, ...);
- **функциональным** – по назначению, исполняемым функциям (описательные, логические, математические);
- **уровню языка** – то есть уровню обобщения в словах-операторах языка (низкого, среднего, высокого, ...);
- **области применения** – системные, сетевые, встроенные и пр.

Поколения языков (Generations of Languages).

Классификация языков по поддерживаемым методологиям появилась примерно в 80-х годах XX века. Каждое из последующих поколений по своей функциональной мощности качественно отличается от предыдущего.

- 1GL - первое поколение: **Машинные языки**. Появились в середине 40-х годов XX века.
- 2GL - второе поколение: **Ассемблеры**. Фактически это те же машинные языки, но более красиво «обернутые». Появились в конце 50-х годов XX века
- 3GL - третье поколение: **Процедурные языки**. Появились в начале 60-х годов XX века. К этому поколению относят универсальные языки высокого уровня, с помощью которых можно решать задачи из любых областей (например, Algol-60).
- 4GL - четвертое поколение: **Языки поддержки сложных структур данных** (например, SQL). Появились в конце 60-х годов XX века.
- 5GL - пятое поколение: **Языки искусственного интеллекта** (например, Prolog). Появились в начале 70-х годов XX века.
- 6GL - шестое поколение: **Языки нейронных сетей** (самообучающиеся языки). Исследовательские работы в этой области начались в середине 80-х годов XX века.

Стиль (парадигма) программирования определяет базовые концепции языков программирования и их сочетания.

Методология включает в себя модель вычислений для данного стиля.

Методология разработки программного обеспечения – совокупность методов, применяемых на различных стадиях жизненного цикла программного обеспечения.

Стили программирования естественно классифицируются по трем признакам:

- низкоуровневые языки;
- высокоуровневые языки;
- глобальность либо локальность действий и условий.

Неструктурное программирование характерно для наиболее ранних языков программирования. Сложилось в середине 40-х с появлением первых языков программирования.

Основные признаки:

- строки как правило нумеруются;
- из любого места программы возможен переход к любой строке;



В основе императивного программирования находятся два основных понятия:

- алгоритм (отсюда название алгоритмические языки)
- архитектура ЭВМ, основанная на модели вычислений фон Неймана.

Допускается выполнение действий над данными определённого типа.

- используются допустимые для данного типа операции и функции:
 - арифметические;
 - логические;
 - символьные;
 - строковые.

Управление ходом исполнения алгоритма.

- используются операторы, реализующие основные управляющие структуры:
 - следование;
 - ветвление;
 - цикл;
 - вызов подпрограммы (возможно).

Декларативное программирование (лат. *declaratio* – объявление, подход возник в 60-х годах) – это предварительная реализация «решателя» для целого класса задач.

Тогда для решения конкретной задачи этого класса достаточно декларировать в терминах данного языка только её условие:

(исходные данные + необходимый вид результата)

«Решатель» сам выполняет процесс получения результата, реализуя известный ему алгоритм решения.

Пример. Найти сумму элементов массива.

Псевдокод:

```
// императивная парадигма
let array = [1, 2, 3, 4, 5, 6];
let result = 0
for (let i = 0; i < array.length; i++)
    result += array[i];
```

Псевдокод:

```
// декларативная парадигма
let array = [1, 2, 3, 4, 5, 6];
array.reduce((prev, current) =>
    prev + current, 0)
```

В **императивной** парадигме разработчик пишет для компьютера инструкции, которым тот следует. Инструкции будут примерно следующие:

- определить массив *array* из 6 элементов с заданными значениями;
- определить переменную *result* и присвоить ей значение 0;
- определить индекс цикла *i* со значением 0;
- начало цикла;
- добавить к переменной *result* элемент массива *array[i]*;
- прибавить к переменной *i* единицу;
- повторять цикл, пока значение переменной *i* меньше количества элементов массива *array*;
- конец цикла;

То есть, программист говорит, что нужно сделать (программирование от глаголов) и в каком порядке, а компьютер выполняет приказы.

В **декларативной** парадигме программист просто пишет следующее:

- дан массив *array* из 6 элементов с заданными значениями;
- получить сумму элементов массива *array*.

Приведенный в примере метод используется для последовательной обработки каждого элемента массива с сохранением промежуточного результата. Здесь *prev* – последний результат вызова функции (промежуточный результат). *current* – текущий элемент массива, элементы перебираются по очереди слева-направо.

При первом запуске *prev* – исходное значение, с которого начинаются вычисления и оно равно нулю.

V. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ



Термин структурное программирование ввёл Э.Дейкстра в 1975 году:

Э.Дейкстра «Заметки по структурному программированию» (в составе сборника «Структурное программирование» / М.: Мир, 1975.

Структурное программирование – стиль написания программ без *goto*.

В структурном программировании необходимо:

- составить правильную логическую схему программы;
- реализовать ее средствами языка программирования;

Программа – множество вложенных блоков (иерархия блоков), каждый из которых имеет один вход и один выход.

Передача управления между блоками и операторами внутри блока (на каждом уровне дерева) выполняется последовательно.

Технология структурного программирования – нисходящее проектирование, т.е. от общего к частному, от внешней конструкции к внутренней.

Теоретическим фундаментом структурного программирования является теорема о структурировании Бёма-Якопини.

Структурное программирование характеризуется:

- ограниченным использованием условных и безусловных переходов;
- широким использованием подпрограмм и управляющих структур (циклов, ветвлений, и т.п.);
- блочной структурой.

Любая вычислимая конструкция может быть представлена комбинацией трёх управляющих структур:

- последовательности;
- ветвления;
- повторения (итерации).

Последовательность – это последовательное выполнение инструкций/блоков.

Ветвление – это выполнение либо одной, либо другой инструкции/блока в зависимости от значения некоего булева (логического) выражения.

Итерация – это многократное выполнение инструкции/блока пока некое булево выражение истинно.

Структурное программирование – методология и технология разработки программных средств, основанная на трёх базовых конструкциях:

- следование;
- ветвление;
- цикл.

Принципы разработки:

- программирование «сверху-вниз» (нисходящее программирование);
- модульное программирование с иерархическим упорядочением связей между модулями/подпрограммами «От общего к частному»

Этапы проектирования:

- формулировка целей (результатов) работы программы;
- представление процесса работы программы (модель);
- выделение из модели фрагментов: определение переменных и их назначения, стандартных программных контекстов.

Технология структурного программирования базируется на следующих методах:

- нисходящее проектирование;
- пошаговое проектирование;
- структурное проектирование (программирование без goto);
- одновременное проектирование алгоритма и данных;
- **модульное проектирование;**
- **модульное, нисходящее, пошаговое тестирование.**

Нисходящее проектирование программы состоит в процессе формализации от самой внешней синтаксической конструкции алгоритма к самой внутренней; в движении от общей формулировки алгоритма к частной формулировке, составляющей его действия;

Структурное проектирование заключается в замене словесной формулировки алгоритма на одну из синтаксических конструкций – *последовательность, условие* или *цикл*.

Использование оператора безусловного перехода **goto** запрещается из принципиальных соображений;

Пошаговое проектирование состоит в том, что на каждом этапе проектирования в текст программы вносится только одна конструкция языка.

Одновременное проектирование алгоритма и структур данных. При нисходящей пошаговой детализации программы необходимые для работы структуры данных и переменные появляются по мере перехода от неформальных определений к конструкциям языка, то есть процессы детализации алгоритма и данных идут параллельно.

Цель (результат) = действие + цель (результат) вложенной конструкции.

Последовательность действий, связанных результатом:

- представить действие в виде последовательности шагов;
- между различными шагами существуют связи через общие переменные, результат выполнения шага и последующие шаги используют этот результат.

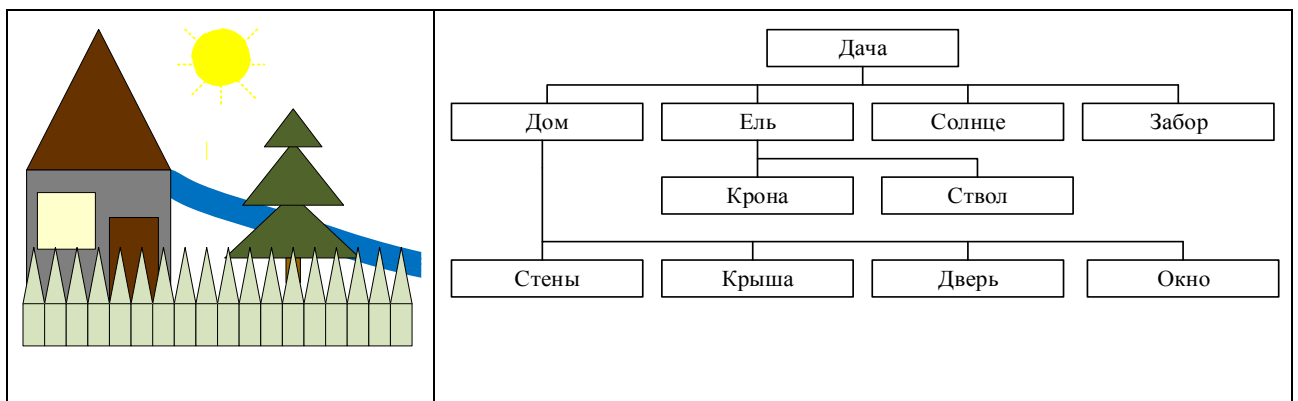
Программирование без goto.

Операторы continue, break и return: служат для более «мягкого» нарушения структурированной логики выполнения программы:

- **continue** – переход завершающую часть цикла;
- **break** – выход из внутреннего цикла;
- **return** – выход из текущего модуля (функции).

VI. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

«Разделяй и властвуй» - латинская формулировка принципа, лежащего в основе модульного проектирования.



Модульное программирование – это организация программы как совокупности небольших независимых блоков, называемых модулями.

Модуль – функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом.

Функциональная декомпозиция задачи – разбиение большой задачи на ряд более мелких, функционально самостоятельных подзадач – модулей.

Каждый модуль в функциональной декомпозиции представляет собой «черный ящик» с одним входом и одним выходом.

Модуль – это фрагмент описания процесса, оформленный как самостоятельный программный продукт, пригодный для многократного использования.

Цели модульного программирования: уменьшить сложность программ; предотвратить дублирование кода, упростить тестирование программы и обнаружение ошибок.

Плюсы модульного программирования:

- ускорение разработки (позволяет изменять реализацию функциональности модуля, не затрагивая при этом взаимодействующие с ним модули);
- повышение надежности (локализует влияние потенциальных ошибок рамками модуля);
- упрощение тестирования и отладки;
- взаимозаменяемость.

Минусы модульного программирования:

- модульность требует дополнительной работы программиста и определенных навыков проектирования программ.

Модуль, основные характеристики:

- один вход и один выход (на вход программный модуль получает набор исходных данных, выполняет их обработку и возвращает набор выходных данных);
- функциональная завершенность (модуль выполняет набор определенных операций для реализации каждой отдельной функции, достаточных для завершения начатой обработки данных);
- логическая независимость (результат работы данного фрагмента программы не зависит от работы других модулей);
- слабые информационные связи с другими программными модулями (обмен информацией между отдельными модулями должен быть минимален);
- размер и сложность программного элемента должна быть в разумных рамках.

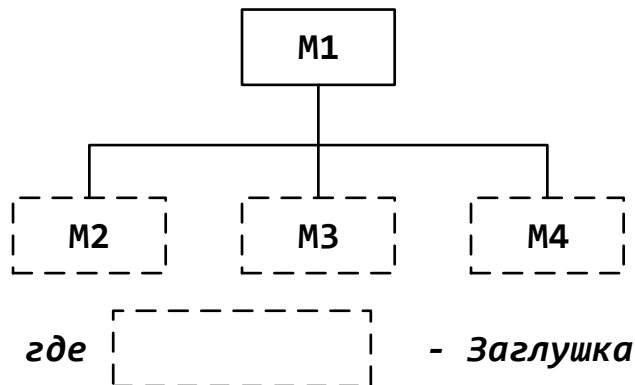
Программный модуль является самостоятельным программным продуктом. Это означает, что каждый программный модуль разрабатывается, компилируется и отлаживается отдельно от других модулей программы.

Роль модулей могут играть структуры данных, библиотеки функций, классы, сервисы и другие программные единицы, реализующие некоторую функциональность и предоставляющие интерфейс к ней.

Технология модульного программирования базируется на следующих методах:

- методы нисходящего проектирования (назначение – декомпозиция большой задачи на меньшие так, чтобы каждую подзадачу можно было рассматривать независимо.);
- методы восходящего проектирования.

Нисходящее проектирование программ и его стратегии.



На первом этапе разработки кодируется, тестируется и отлаживается головной модуль, который отвечает за логику работы всего программного комплекса. Остальные модули заменяются заглушками, имитирующими работу этих модулей.

На последних этапах проектирования все заглушки постепенно заменяются рабочими модулями.

Стратегии, на которой основана реализация:

- пошаговое уточнение (данная стратегия разработана Е. Дейкстрой);
- анализ сообщений (данная стратегия базируется на работах группы авторов: Йодана, Константайна, Мейерса).

Пример. Стратегия, основанная на использовании псевдокода.

Способы реализации пошагового уточнения.

1. Кодирование программы с помощью псевдокода и управляющих конструкций структурного программирования;
2. Использование комментариев для описания обработки данных. Пошаговое уточнение требует, чтобы взаимное расположение строк программы обеспечивало читабельность всей программы.

Преимущества метода пошагового уточнения:

- основное внимание при его использовании обращается на проектирование корректной структуры программ, а не на ее детализацию;
- так как каждый последующий этап является уточнением предыдущего лишь с небольшими изменениями, то легко может быть выполнена проверка корректности процесса разработки на всех этапах.

Недостаток метода пошагового уточнения:

- на поздних этапах проектирования может возникнуть необходимость в структурных изменениях, которые повлекут за собой пересмотр более ранних решений.

Использование псевдокода.

Пример.

Пусть программа обрабатывает файл с датами.

Необходимо отделить правильные даты от неправильных, отсортировать правильные даты, определить летние даты и вывести их в выходной файл.

Первый этап пошагового уточнения: **задается** заголовок программы, соответствующий ее основной функции:

Program Обработка_дат

Второй этап пошагового уточнения: **определяются** основные действия.

Program Обработка_дат;

Отделить_правильные_даты_от_неправильных {*}

Обработать_неправильные_даты;

Сортировать_правильные_даты;

Выделить_летние_даты;

Вывести_летние_даты_в_файл;

EndProgram.

Третий этап пошагового уточнения: **детализация** фрагмента *.

Program Обработка_дат;

While не_конец_входного_файла

Do

Begin

Прочитать_дату;

Проанализировать_правильные|неправильные_даты;

End

Обработать_неправильные_даты;

Сортировать_правильные_даты;

Выделить_летние_даты;

Вывести_летние_даты_в_файл;

EndProgram.

и так далее.

Метод восходящей разработки

При восходящем проектировании разработка идет снизу-вверх.

На первом этапе разрабатываются модули самого низкого уровня.

На следующем этапе к ним подключаются модули более высокого уровня и проверяется их работоспособность.

На завершающем этапе проектирования разрабатывается головной модуль, отвечающий за логику работы всего программного комплекса.

Методы нисходящего и восходящего программирования имеют свои преимущества и недостатки.

Недостатки нисходящего проектирования:

- необходимость заглушек;
- до самого последнего этапа проектирования неясен размер всего программного комплекса и его характеристики, которые определяются только после реализации модулей самого низкого уровня.

Преимущество нисходящего проектирования – на самом начальном этапе проектирования отлаживается головной модуль (логика программы).

Недостаток восходящего программирования – головной модуль разрабатывается на завершающем этапе проектирования, что порой приводит к необходимости дорабатывать модули более низких уровней.

Преимущество восходящего программирования – не нужно писать заглушки.

На практике применяются оба метода. Метод нисходящего проектирования чаще всего применяется при разработке нового программного комплекса, а метод восходящего проектирования – при модификации уже существующего комплекса.

VII. СТАНДАРТ ОФОРМЛЕНИЯ КОДА

Стандарт оформления кода – набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования.

Стандарт оформления кода (или стандарт кодирования, или стиль программирования) (англ. coding standards, coding convention или programming style).

Это набор соглашений, который принимается и используется некоторой группой разработчиков программного обеспечения для единообразного оформления совместно используемого кода.

Целью принятия и использования стандарта является упрощение восприятия программного кода человеком, минимизация нагрузки на память и зрение при чтении программы.

Стандарт оформления кода описывает:

- способы выбора названий и используемый регистр символов для имен переменных и других идентификаторов:
 - запись типа переменной в ее идентификаторе;
 - регистр символов (нижний, верхний, «верблюжий», «верблюжий» с малой буквы), использование знаков подчёркивания для разделения слов;
- стиль отступов при оформлении логических блоков – используются ли символы табуляции, ширина отступа;
- способ расстановки скобок, ограничивающих логические блоки;
- использование пробелов при оформлении логических и арифметических выражений;
- стиль комментариев и использование документирующих комментариев.

Вне стандарта подразумевается:

- отсутствие магических чисел;
- ограничение размера кода по горизонтали (чтобы помещался на экране) и вертикали (чтобы весь код файла держался в памяти);
- ограничение размера функции или метода – в размер одного экрана.