МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Go For IT English Reading

УЧЕБНОЕ ПОСОБИЕ
по английскому языку
для бакалавров 1−2 курса
факультета информационных систем
и технологий очной формы обучения

Составители: Л. В. Корухова
Н. Н. Новосельцева

Ульяновск
УлГТУ
2016

УДК 811.11(075.8)
ББК 81.2 Англ я7
G 69

Рецензенты: Кафедра германской филологии Черниговского национального педагогического университета им. Т. Г. Шевченко
Морозова М. А., канд. пед. наук, доцент кафедры иностранных языков Ульяновского института гражданской авиации им. главного маршала авиации Б. П. Бугаева

Утверждено редакционно-издательским советом университета
в качестве учебного пособия

G 69 **Go For IT English Reading** : учебное пособие по английскому языку для бакалавров 1−2 курса факультета информационных систем и технологий очной формы обучения / сост. Л. В. Корухова, Н. Н. Новосельцева. – Ульяновск : УлГТУ, 2016. – 168 с.

ISBN 978-5-9795-1608-0

Пособие разработано в соответствии с требованиями стандарта высшего профессионального преподавания и ориентировано на обучение студентов профессионально-ориентированному английскому языку в рамках курса. Материал пособия, основанный на современных аутентичных текстах зарубежных авторов, представленный в основном блоке, дает возможность форми-рования и совершенствования навыков чтения и перевода текстов по специальности на иностранном языке, усвоению базового терминологического корпуса.

Работа подготовлена на кафедре «Иностранные языки» УлГТУ.
Печатается в авторской редакции.

УДК 811.11(075.8)
ББК 81.2 Англ я7

ISBN 978-5-9795-1608-0

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Дополнительное домашнее чтение или внеаудиторное чтение – вид учебной деятельности, который требует от студента самостоятельности. Эта самостоятельность проявляется не только в организации работы, но и в выборе текста, который, в свою очередь, и вызывает определенную сложность. Тематика текстов по дополнительному домашнему чтению – профессионально-ориентированная. Материалы профессионально-ориентированной тематики – это тексты, тесно связанные со специальностью. Как отмечают многие исследователи, чтение текстов на английском языке – один из способов изучения иностранного языка. При правильном подборе оригинальные тексты могут способствовать формированию языковой компетенции специалиста. В этой связи для эффективной организации самостоятельной работы студентов 1–2 курсов факультета информационных систем и технологий по внеаудиторному чтению было составлено учебное пособие, включающее аутентичные профессионально-ориентированные тексты.

Главной целью обучения чтению литературы по специальности в неязыковом вузе является получение информации из текстов на иностранном языке. Чтение литературы на иностранном языке позволяет современному специалисту научиться самостоятельно работать с иноязычными научно-техническими текстами и своевременно получать новую информацию в области своей специальности. Большинство студентов также признают необходимость использования иностранной литературы при подготовке курсовых проектов, дипломных работ, при работе в Интернете. Кроме того, планомерное домашнее чтение является важным средством увеличения словарного запаса и развития навыков устной речи.

Внеаудиторное чтение как самостоятельный аспект учебной дисциплины «Иностранный язык» способствует более прочному формированию всех видов коммуникативной компетенции.

Основными функциями внеаудиторного чтения являются: образовательная, развивающая, воспитательная и практическая:

- Образовательная функция вносит существенный вклад в повышение образования студентов, расширяет кругозор.
- Развивающая функция способствует развитию критического мышления, когнитивных способностей студентов, ценностных ориентиров.

- Воспитательная функция внеаудиторного чтения заключается в воспитании студентов в духе мира, толерантности, гуманного межнационального общения.
- Практическая функция внеаудиторного чтения заключается в развитии коммуникативных умений чтения как особого вида человеческой деятельности.

Внеаудиторное чтение оказывает существенное воздействие на личность обучаемого, его интеллектуальную, эмоциональную и мотивационные сферы и предполагает различного вида работу с текстом (интерпретацию, соотнесение содержания со своим личным опытом, умение аргументированно изложить понимание проблемы и др.).

ЧАСТЬ 1: О НАУЧНО-ТЕХНИЧЕСКОМ ПЕРЕВОДЕ

Перевод − это выражение того, что уже было выражено на одном языке средствами другого языка. Суть технического перевода и основной его особенностью является полное понимание вопроса и объяснение его доступным языком с соблюдением всех особенностей и технических тонкостей материала, который подлежит переводу.

Перевод − искусство, требующее умения рассредоточить внимание настолько, чтобы, занимаясь частностями, всегда иметь в виду целое, т. е. весь текст. Нельзя изолированно переводить слова, фразу за фразой, предложение за предложением, т. е. нельзя делать то, что называется буквальным переводом. Отдельные слова вне контекста не имеют смысла, т. к. слова многозначны. Неумение отвлечься от конкретных форм и неумение пользоваться контекстом − типичная ошибка при переводе.

Непереводимых оригиналов нет, т. к. то, что можно выразить на одном языке, можно выразить и на любом другом. Есть только трудности, связанные с недостаточными знаниями либо языка, либо существа предмета.

Следует отметить следующие особенности стиля научно-технического перевода: информационная содержательность, строгая последовательность и логичность всех составляющих представляемой автором идеи, объективность в констатации научного факта, понятность излагаемого материала в первую очередь для специалиста в этой области. Научно-технический текст прежде всего отличается существом излагаемой информации, где особая роль принадлежит терминологии, используемой в процессе перевода. Поскольку научный термин отражает определенное научное понятие, то этот термин отличается от обычных слов общенационального языка, но обязательно должен отражать реальные объекты и явления и устанавливать однозначное понимание явления специалистами в этой области. Поэтому термин, используемый при переводе, должен быть точным, иметь строго определенное значение и входить в так называемое понимаемое читателем «терминологическое поле», быть элементом общей терминологической системы при объяснении существа исследуемого вопроса.

Термин − это слово или словосочетание, которое может иметь отличное от обиходного значение в зависимости от области науки и техники, в которой оно употребляется. Термин может быть

простым, состоящим из одного слова (*switch* «выключатель») и сложным термином-словосочетанием (*automatic switch* «автоматический выключатель», *high-speed circuit breaker* «быстродействующий выключатель») Например, слово *face* как существительное имеет обиходное значение «лицо»; широкое техническое значение его – «поверхность»; в геометрии его значение – «грань»; в строительном деле – «фасад», «облицовка»; в программировании – «стиль». Но, как это естественно для английского языка, то же слово *face* может выступать как глагол; в этом случае его основное значение (обиходное значение) – «стоять напротив чего-либо»; в металлообработке это термин означает «шлифовать», в строительном деле – «отделывать», «облицовывать», «покрывать».

Соблюдение следующих правил может помочь при переводе терминов.

В специальном тексте каждое слово, даже очень хорошо знакомое, может оказаться термином. Переводя техническую литературу, особенно по малознакомой тематике, надо всегда помнить об этой многозначности. Пользуйтесь при переводе специальными техническими словарями. Следует считать неразумной попытку переводить без словаря; это выглядело бы так же странно, как если бы мастер пытался научиться работать без инструмента. Как бы ни была велика память переводчика, он может натолкнуться на незнакомый или малознакомый термин или известный ему термин в совершенно новом значении. При многозначности слова следует брать то его значение, которое принадлежит соответствующей области техники. Может оказаться, что ни одно из значений слова, найденных вами в словаре, не подходит, это значит, что некоторые значения слова не зафиксированы в словаре. В таком случае вывести из затруднения может хорошее чувство языка, так называемая языковая догадка, но в первую очередь − понимание того, о чем идет речь. Поэтому знакомство с соответствующей отраслью техники, хотя бы по популярным пособиям или с помощью консультации специалиста, имеет огромное значение для правильного перевода.

Для создания точного и последовательного процесса перевода необходимо следовать нескольким несложным правилам:

1. Первый раз необходимо внимательно прочитать текст без словаря и попытаться понять его. Помните, замысел автора выражен с помощью всего текста.

2. Прочитайте текст второй раз по отдельным предложениям, попытайтесь понять синтаксический строй и смысл каждого предложения. Переведите текст по предложениям.

3. Если синтаксический строй предложения Вам неясен и Вы не поняли смысл предложения, сделайте грамматический анализ:

- определите вид предложения;
- найдите подлежащее, сказуемое, второстепенные члены;
- выделите в предложении смысловые группы, т. е. поделите предложение на смысловые отрезки, помня, что правила пунктуации в русском и английском языке различны, и чаще всего смысловые отрезки в английском предложении не будут отделены запятой или другим знаком препинания.
- если предложение сложноподчиненное, найдите главное и придаточное предложения, опираясь на формальные признаки.

4. Выписать и перевести все незнакомые слова, по ходу перевода текста, учитывая контекст и части речи.

5. Переводить текст, помня об особенностях русского научно-технического стиля. Помните, при переводе последующего предложения необходимо постоянно удерживать в памяти смысл предыдущего, иначе теряется логическая связь между отдельными предложениями.

Необходимо постоянно следить за тем, чтобы между каждой последующей и предыдущей частью перевода была логическая связь.

При переводе научно-технических текстов необходимо соблюдать:

- Точность. Обязательно должен сохраняться смысл оригинала, сам перевод не должен содержать непереведенных аббревиатур, непереведенных слов или пропусков.
- Терминология. Должно соблюдаться единство терминологии на протяжении всего текста. Например, если в начале текста часть корпуса какого-то агрегата названа «дном», то в дальнейшем нельзя называть ее «основанием», «днищем» и т. п.
- Обязательное соблюдение языковых норм. При переводе не должны допускаться грамматические ошибки, ошибки в управлении, синтаксисе или согласовании времени. Все слова должны употребляться в правильном порядке. Нельзя нарушать лексические нормы, допускать орфографические ошибки и опечатки.
- Стиль. Существующая **особенность научно-технического перевода** не должна нарушать стилистическое единство изложенного.

Стиль тематики должен соответствовать области применения перевода. Не должны встречаться слова-паразиты, необоснованные повторы, а сам текст должен восприниматься легко. При переводе должна сохраняться синтаксическая структура оригинала.

ЧАСТЬ 2: ФРАЗЫ ДЛЯ ПЕРЕСКАЗА (РЕФЕРИРОВАНИЯ) ТЕКСТА

The beginning:

The article / paper / book	***deals with…***	– Эта статья / работа / книга
	is concerned with…	касается /
	is devoted to…	посвящена

As the title implies the article describes… – Согласно названию, в статье описывается…

The text/article is about ... – Текст / Статья – о…

The text/article deals with the problem of (the issue of)... – Текст/Статья касается проблемы…

The contents of the article:

According to the article / text… – Согласно статье/тексту…

It is specially noted that… – Особенно отмечается…

A mention should be made about… – Упоминается…

It is spoken in detail about… – Подробно описывается…

…is / are noted – Упоминается / Упоминаются…

One of the main points to be singled out is... – Одна из главных идей, которую нужно подчеркнуть, – это…

It is reported that… – Сообщается…

It is shown that… – Показано, что…

The text gives a valuable information on... / on the fact that… – Текст дает ценную информацию…

Much attention is given to… / to the fact that… – Большое внимание уделяется…

It gives a detailed (thorough) analysis of… – В статье подробно анализируется...

It draws our attention to…/to the fact that… – Статья привлекает наше внимание к …

The difference between the terms … and … should be stressed – Следует подчеркнуть различие между терминами … и …

It should be stressed / emphasized / noted that… – Следует подчеркнуть, что…

…is/are proposed – Предлагается / Предлагаются…

…is / are examined – Проверяется / Исследуются…

The conclusion:

From what the author says it becomes clear that… – Из слов автора становится ясно, что…

The article is of great help to… – Эта статья окажет большую помощь…

The article is of interest to… – Эта статья представляет интерес для…

I think / In my opinion / I found the article (rather) interesting (important, useful) as / because… – Я считаю статью (довольно) интересной (важной, полезной), потому что …

ЧАСТЬ 3: ТЕКСТЫ ДЛЯ ВНЕАУДИТОРНОГО ЧТЕНИЯ

TEXT 1 OPERATING SYSTEM

Functions of an Operating System

The operating System's (OS) function is to manage the main components of a computer and act as a user interface for the computer's hardware. The OS plays an important role for the entire computer system. The operating system is responsible for performing the following activities, provides a user interface, performs common hardware functions, manages system memory, manages processing tasks, provides network capability, controls access to system resources and manages files.

Types of system interfaces

Providing a system interface, which allows users the ability to access the computer system, is a principle function of any operating system. There exist many different types of system interfaces, which include command-based user interfaces and graphical user interfaces.

Command-based user interface

The first computer system interfaces were command based. A command-based user interface requires users to memorize commands and type them in order to run programs and accomplish tasks. Such user

interfaces were predominately in personal computers used until Windows 3.1 became standard issue in 1992.

Graphical user interface

The main difference between the graphical user interface and the command-based user interface is that the prior uses icons, menus and button-bars, which are activated by mouse to operate software. In his writings, Cardinali (1994) explains that graphical user interfaces are linked to an increase in productivity amongst users. Studies shown that the graphical user interface significantly reduces the learning curve opposed to it's command driven counterpart. Cardinali (1994) attributes the graphical interface's preference among users to the elimination of having to memorize commands. The most commonly used graphical user interface, today, is Windows by Microsoft.

Common hardware functions

The performance of a computer depends on the Operating system's management of hardware, which includes extrapolating data from input devices or retrieving data from disks, storing the data and displaying the information via output devices such as a monitor. The OS converts simple instructions for the tasks above into detailed instructions that the computer is able to interpret. In addition the OS communicates errors and attention needs required by input/output devices.

Memory Management and processing tasks

Sobh and Tibrewal (2006) state, «Memory is an important resource that must be carefully managed». The memory management feature is responsible for directing user requests for date to the data's physical storage location. Other functions of the memory management feature include space multiplexing and multitasking. Space multiplexing means that more than one user can be operating the OS at the same time, under this assumption, the OS schedules every process in such a way that users get the impression that their processes reside directly on the RAM. Multitasking allows users to run more than one application simultaneously. In short the job of the memory manager is to keep track of which parts of memory are in use and which parts are not in use, to allocate memory to processes when they need it and de-allocate it when they are done, and to manage swapping between main memory and disc when main memory is not big enough to hold all the processes.

Networking capability, system resources, and file management

Some operating systems provide features that allow users to connect to computer networks and the capability to link users to the Internet (Stair

and Reynolds, 2006). The networking capability of the OS makes a user more vulnerable to security issues. Therefore, the OS is equipped with protection features such as password protected log-on features, the recording of user information pertaining to the log-on, and the reporting of security breaches. Furthermore, users may require that more than one person have access to the same computer. The OS protects multiple users on the same computer by keeping track of where each file is stored and who is authorized to access it.

Market share and trends

- The major operating systems are Windows, Mac OS, UNIX and Linux.
- Windows decisively dominates the workstation market, with almost 90% of market share. Mac OS has recently increased its share to about 10%.
- Windows XP is still the major workstation operating system, with 65% to Vista's 24%.
- Windows and UNIX each command about a third of the server market.

Pros, Cons and Costs

- The major advantages of Windows are availability of business applications software, support and resources, and very high mainstream comfort levels.
- The high cost of matching Windows applications on other systems and the high cost of migration are the greatest barriers to change.
- Despite somewhat higher hardware costs, Mac TCO is lower than that of Windows.
- Mac OS offers the advantages of less «user friction interface», higher productivity, lower maintenance and support costs, and better security.
- Windows licensing costs are far higher than the other operating systems. Mac servers include unlimited licences, and UNIX and Linux are free.
- Estimates were that there would be 500,000 new pieces of malware (viruses, trojans etc) for Windows by the end of 2008. Threats to Mac OS, UNIX and Linux are negligible.
- TCO findings for servers are mixed, with some studies finding higher TCO for UNIX and Linux, others for Windows – depending on who commissioned the study.
- Linux and UNIX offer low maintenance requirements, exceptional stability and very good security.

▪ Mixed networks often make tremendous business sense, as they allow businesses to match systems to their needs instead of vice versa.

Conclusion

The Operating system is responsible for providing users with an interface that allow them to communicate directions in order to perform specified tasks on a computer. The OS facilitates the above process by managing common hardware functions, where it converts simple directions into detailed instructions the computer can interpret. Furthermore, the OS manages memory and processing tasks, which allow users to store, and request data, to run more than one application at a time and even allow multiple users access to the same system. Finally, the OS secures the users files by managing its resource allocation and files, which allow users to comfortably share their computer and networking capabilities with others.

(5,242 symbols)

http://www.nashnetworks.ca/pros-cons-and-costs-of-operating-systems.htm

https://jennadoucet.wordpress.com/2010/03/14/functions-of-an-operating-system/

TEXT 2 29 YEARS OF WINDOWS EVOLUTION

Microsoft's Windows operating system was first introduced in 1985. Over 29 years later a lot has changed, but what things have stayed the same? Microsoft Windows has seen nine major versions since its first release in 1985. Over 29 years later, Windows looks very different but somehow familiar with elements that have survived the test of time, increases in computing power and – most recently – a shift from the keyboard and mouse to the touchscreen.

Here's a brief look at the history of Windows, from its birth at the hands of Bill Gates with Windows 1 to the latest arrival under new Microsoft chief executive Satya Nadella.

Windows 1

This is where it all started for Windows. The original Windows 1 was released in November 1985 and was Microsoft's first true attempt at a graphical user interface in 16-bit.

Development was spearheaded by Microsoft founder Bill Gates and ran on top of MS-DOS, which relied on command-line input.

It was notable because it relied heavily on use of a mouse. To help users become familiar with this odd input system, Microsoft included a game, Reversi that relied on mouse control, not the keyboard, to get people used to moving the mouse around and clicking onscreen elements.

Windows 2

Two years after the release of Windows 1, Microsoft's Windows 2 replaced it in December 1987. The big innovation for Windows 2 was that windows could overlap each other, and it also introduced the ability to minimise or maximise windows instead of «iconising» or «zooming».

The control panel, where various system settings and configuration options were collected together in one place, was introduced in Windows 2 and survives to this day.

Microsoft Word and Excel also made their first appearances running on Windows 2.

Windows 3

The first Windows that required a hard drive launched in 1990. Windows 3 was the first version to see more widespread success and be considered a challenger to Apple's Macintosh and the Commodore Amiga graphical user interfaces.

Windows 3 introduced the ability to run MS-DOS programmes in windows, which brought multitasking to legacy programmes. Windows 3 supported 256 colours bringing a more modern, colourful look to the interface.

Windows 3.1

Windows 3.1 released in 1992 is notable because it introduced TrueType fonts making Windows a viable publishing platform for the first time.

Minesweeper also made its first appearance. Windows 3.1 required 1MB of RAM to run and allowed supported MS-DOS programs to be controlled with a mouse for the first time. Windows 3.1 was also the first Windows to be distributed on a CD-ROM.

Windows 95

As the name implies, Windows 95 arrived in August 1995 and with it brought the first ever Start button and Start menu. New windows launched with a gigantic advertising campaign.

It also introduced the concept of "plug and play" – connect a peripheral and the operating system finds the appropriate drivers for it and makes it work. That was the idea; it didn't always work in practice.

Windows 95 also introduced a 32-bit environment, the task bar and focused on multitasking. MS-DOS still played an important role for Windows 95, which required it to run some programmes and elements. Internet Explorer also made its debut on Windows 95, but was not installed by default.

Windows 98

Released in June 1998, Windows 98 built on Windows 95 and brought with it IE 4, Outlook Express, Windows Address Book, Microsoft Chat and NetShow Player, which was replaced by Windows Media Player 6.2 in Windows 98 Second Edition in 1999.

Windows 98 introduced the back and forward navigation buttons and the address bar in Windows Explorer, among other things. One of the biggest changes was the introduction of the Windows Driver Model for computer components and accessories – one driver to support all future versions of Windows.

USB support was much improved in Windows 98 and led to its widespread adoption, including USB hubs and USB mice.

Windows ME

Windows ME was the successor to Windows 98 SE and was targeted specifically at home PC users. It included Internet Explorer 5.5, Windows Media Player 7, and the new Windows Movie Maker software, which provided basic video editing and was designed to be easy to use for users. Microsoft also updated the graphical user interface.

Windows 2000

Windows 2000 was released in February 2000 and was based on Microsoft's business-orientated system Windows NT and later became the basis for Windows XP.

Microsoft's automatic updating played an important role in Windows 2000 and became the first Windows to support hibernation.

Windows XP

Arguably one of the best Windows versions, Windows XP was released in October 2001.

It was based on Windows NT like Windows 2000, but brought the consumer-friendly elements from Windows ME. The Start menu and task bar got a visual overhaul, bringing the familiar green Start button, blue task bar and vista wallpaper and other visual effects.

ClearType, which was designed to make text easier to read on LCD screens, was introduced, as were built-in CD burning, autoplay from CDs and other media, plus various automated update, that unlike Windows ME actually worked.

Windows XP was the longest running Microsoft operating system, seeing three major updates and support up until April 2014 – 13 years from its original release date. Windows XP was still used on an estimated 430m PCs when it was discontinued.

Its biggest problem was security: though it had a firewall built in, it was turned off by default. Windows XP's huge popularity turned out to be a boon for hackers and criminals, who exploited its flaws.

Windows Vista

Windows XP to remain competitive for six years before being replaced by Windows Vista in January 2007. Vista updated the look and feel of Windows with more focus on security.

It also ran slowly on older computers. PC gamers saw a boost from Vista's inclusion of Microsoft's DirectX 10 technology.

Windows Media Player 11 and IE 7 debuted, along with Windows Defender an anti-spyware programme. Vista also included speech recognition, Windows DVD Maker and Photo Gallery. Later a version of Windows Vista without Windows Media Player was created in response to anti-trust investigations.

Windows 7

Considered by many as what Windows Vista should have been, Windows 7 was first released in October 2009. It was intended to fix all the problems and criticism faced by Vista, with slight tweaks to its appearance and a concentration on user-friendly features.

It was faster, more stable and easier to use. For this reason, users and would upgrade to from Windows XP, forgoing Vista entirely.

Windows 7 continued improvements on Windows Aero (the user interface introduced in Windows Vista) with the addition of a redesigned taskbar that allows applications to be "pinned" to it, and new window management features. Other new features were added to the operating system, including libraries, the new file sharing system HomeGroup, and support for multitouch input.

In contrast to Windows Vista, Windows 7 was generally praised by critics, who considered the operating system to be a major improvement over its predecessor due to its increased performance, its more intuitive interface.

Windows 8

Released in October 2012, Windows 8 was Microsoft's most radical overhaul of the Windows interface, ditching the Start button and Start menu in favour of a more touch-friendly Start screen.

The new tiled interface replaces the lists of programmes and icons. A desktop was still included, which resembled Windows 7.

Windows 8 was faster than previous versions of Windows and included support for the new, much faster USB 3.0 devices. The Windows

Store, which offers universal Windows apps that run in a full-screen mode only, was introduced. Programs could still be installed from third-parties like other iterations of Windows.

The radical overhaul was not welcomed by many. Microsoft attempted to tread a fine line between touchscreen support and desktop users, but ultimately desktop users wanting to control Windows with a traditional mouse and keyboard and not a touchscreen felt Windows 8 was a step back. Even despite the parallel rise of tablets such as the iPad, and smartphones, which had begun outselling PCs by the end of 2010.

A free point release to Windows 8 introduced in October 2013, Windows 8.1 marked a shift towards yearly software updates from Microsoft.

Windows 10

Announced on 30 September 2014, Windows 10 has only been released as a test version for keen users to try. The "technical preview" is very much still a work in progress.

Some interesting features include the ability to switch between a keyboard and mouse mode and a tablet mode, for those computers like the Surface Pro 3 with a detachable keyboard.

Windows 10 – despite being the ninth version of Windows – is designed to unify all Windows platforms across multiple devices, including Windows Phone and tablets, with universal apps that can be downloaded from the Windows Store and run on all Windows devices. Windows 10 was also criticized for limiting how users can't control its operation; in particular, Windows Update installs all updates automatically, no longer allows users to selectively install updates, and only the Pro edition of Windows 10 can delay the automatic installation of new builds of the platform.

(7,401 symbols)

https://www.theguardian.com/technology/2014/oct/02/from-windows-1-to-windows-10-29-years-of-windows-evolution

TEXT 3 UNIX

The uniqueness of UNIX

The features that made UNIX a hit from the start are: multitasking capability, multi-user capability, portability, UNIX programs, library of application software, security.

1. Multitasking Capability

Many computers do just one thing at a time, as anyone who uses a PC or laptop can attest. Try logging onto your company's network while opening your browser while opening a word processing program. Chances are the processor will freeze for a few seconds while it sorts out the multiple instructions. UNIX, on the other hand, lets a computer do several things at once, such as printing out one file while the user edits another file. This is a major feature for users, since users don't have to wait for one application to end before starting another one.

2. Multi-user

The same design that permits multitasking permits multiple users to use the computer. The computer can take the commands of a number of users – determined by the design of the computer – to run programs, access files, and print documents at the same time.

The computer can't tell the printer to print all the requests at once, but it does prioritize the requests to keep everything orderly. It also lets several users access the same document by compartmentalizing the document so that the changes of one user don't override the changes of another user.

3. System portability

A major contribution of the UNIX system was its portability, permitting it to move from one brand of computer to another with a minimum of code changes. At a time when different computer lines of the same vendor didn't talk to each other – yet alone machines of multiple vendors – that meant a great savings in both hardware and software upgrades. It also meant that the operating system could be upgraded without having all the customer's data inputted again. And new versions of UNIX were backward compatible with older versions, making it easier for companies to upgrade in an orderly manner.

4. UNIX Programs

UNIX comes with hundreds of programs that can divide into two classes; Integral utilities These are absolutely necessary for the operation of the computer, such as the command interpreter, and Tools that aren't necessary for the operation of UNIX but provide the user with additional capabilities, such as typesetting capabilities and e-mail.

UNIX Communications

E-mail is commonplace today, but it has only come into its own in the business community within the last 10 years. Not so with UNIX users, who have been enjoying e-mail for several decades. UNIX e-mail at first permitted users on the same computer to communicate with each other via their terminals. Then users on different machines, even made by different

vendors, were connected to support e-mail. And finally, UNIX systems around the world were linked into a world wide web decades before the development of today's World Wide Web.

5. Applications libraries

UNIX as it is known today didn't just develop overnight. Nor were just a few people responsible for it's growth. As soon as it moved from Bell Labs into the universities, every computer programmer worth his or her own salt started developing programs for UNIX. Today there are hundreds of UNIX applications that can be purchased from third-party vendors, in addition to the applications that come with UNIX.

6. Security.

It is safe, preventing one program from accessing memory or storage space allocated to another, and enables protection, requiring users to have permission to perform certain functions, i.e. accessing a directory, file, or disk drive.

UNIX at HOME

A few of the many advantages of using Unix at home are: Unix runs on older, less powerful machines. If your machine does not have enough CPU speed and memory for Windows, it can still run Unix.

Several Unix flavors such as FreeBSD are free. Additionally high quality, free applications like the emacs text editor, Apache web server and GIMP image editor are available for Unix platforms. Equivalent Windows software costs hundreds of dollars

Unix provides a flexible multi-user environment. Each member of the family can have their own account with personal settings and secure files. You can keep the kids from reading your work, financial or personal files, while allowing your spouse to access financial files but not work or personal files.

Unix provides the ultimate in computer programming environments. Powerful C, C++, Fortran and Java compilers along with development tools are available for free. Furthermore, the Internet is littered with libraries of free code for these compilers. It would cost over $1,000 to create a comparable programming environment for a machine running Windows.

On the downside

You should only consider Unix if you are willing to spend a lot of time working on your operating system. Unix is harder to install, maintain and upgrade than Windows or the MacOS. More home oriented applications run under Windows than Unix. If you need educational software for the kids or love computer games, Windows is a better choice than Unix.

Dual booting (running Windows and Unix on the same machine) overcomes this disadvantage.

UNIX Evaluation

In making a decision on which OS is better, we must finally evaluate the abundant advantages of UNIX over the disadvantages. As programs continue to become larger and more complex, and as computers become faster and increase in complexity, operating systems must become more and more stable. In comparison with Windows NT, UNIX maintains this stability very well.

NASA, who relies very heavily on their equipment, prefers UNIX because if its stability in complex, mission-critical tasks as UNIX very rarely crashes (see graph below). When UNIX does actually crash, only parts of it crash so the system is often easily recoverable without rebooting (UNIX-Based 37). In addition, UNIX tends to beat Windows NT in performance tests. Companies and organizations often create performance tests where an operating system is put under an intense load to see how it performs against other operating systems under the same work load. A newly released study shows that Windows NT finished last in comparison with five UNIX versions (See Outside Source: UNIX Trounces Windows NT in Tests). The tests measured reliability, ease of management, stability, and performance among other things (UNIX Trounces).

Finally, but still very important, although UNIX can be more expensive than Windows NT initially, it can actually cost less in the long run. The «base» software in UNIX generally costs more than Windows NT, but client-access licenses and add-on packages often do not. What is a client-access license? Like most published information, programs are restricted by copyright laws. One cannot just legally copy a program from one computer to another without paying for another copy of that software. In the networked environment, servers are licensed in a different way. They are licensed according to how many other computers are trying to use its data at the same time. For example, in a network of seven computers connected together, one being the server, if all six computers were logged in (using the server's data) at the same time, then the server would need six of what are called client-access licenses. Windows NT continually charges more and more for each client-access license added, while you can usually get an unlimited client-access license for UNIX for about $1200. One could spend thousands more in Windows NT client-access licenses. (Microsoft). As well, add-on software packages (such as those that manage e-mail for many users) for UNIX often cost less (and are often already included with UNIX) than those for Windows NT.

Disadvantages of UNIX

In continuing the evaluation of Windows NT and UNIX, I will show you the disadvantages and advantages of UNIX. UNIX generally holds its disadvantages in its interaction with users. As hinted to before, UNIX is not very user-friendly to beginners. Many people, including UNIX gurus (people who know about and often like UNIX), agree that UNIX is not as user-friendly. Using UNIX can often require the knowledge of basic commands as opposed to just using the mouse. X Windows, a system similar to Microsoft Windows that runs on UNIX, is also not as easy and attractive to use as Windows (Showdown). However, this is changing rapidly. An organization that produces free software, known as the KDE Free Qt Foundation, has, in fact, developed a windowing system called the K Desktop Environment that many people consider to be far superior to the Windows NT interface in usability, customizability, and stability (Systems 34).

Finally, users do not have as large of a choice for programs under UNIX as they do under Windows. A program developed for the Microsoft Windows environment cannot automatically be run on UNIX. Some modifications need to be made to a program before it can be run on a different operating system other than for which it was intended to. Companies that develop software have been reluctant to develop programs for UNIX because not as many people use the operating system. However, as Linux, a free operating system that greatly resembles and is almost comparable to UNIX (Reborn), is increasing in popularity, companies are continually providing more and more programs for UNIX (Showdown).

Different Unix systems

They are many different versions of Unix, as well as some Unix 'lookalikes'. The most widely used are: • System V (distributed by the original developers, AT&T) • AIX (IBM) • Berkeley BSD (from the University of California, Berkeley) • SunOS, now known as Solaris (from the makers of Sun workstations) • Xenix (a PC version of Unix).

(7,802 symbols)

http://www.123helpme.com/unix-operating-system-view.asp?id=159727

TEXT 4 LINUX

The term “Linux” is used to describe Open Source software, consisting of an operating system kernel, a graphical user environment, software configuration, utilities and applications that make a computer

usable. There are many different versions of Linux available. These versions are referred to as “distributions”.

What Linux distributions, desktop environments and most Linux applications all have in common is: They are Open Source. The Linux community is a type of Open Source community focused on the Linux operating system and its applications, and using, improving and supporting them.

Open Source Community

Using open source software like Linux, and Linux applications provides you with the freedom to run a complete, full-featured operating system, pre-configured with most, if not all, of the applications you will need for your daily computing – or to change anything about the way it looks, the way it works, or the applications it runs to suit your taste. Although you will find some distributions of Linux for purchase, the vast majority are provided free of charge. Open Source software is licensed in a way that allows anyone to give it away for free, no strings attached.

For example, the licence gives any member of the user community the freedom to use Linux for any purpose, to distribute, modify, redistribute, or even sell the operating system. If you do modify and then redistribute Linux with your modifications, you are required by the licence to submit your modifications for possible inclusion into future versions. There is no guarantee that this will ever happen, but if you have made it better, then your changes just might be included in the next release of your distribution of Linux.

Developer Community

Many users of Linux are corporations that use the operating system to run their businesses, or include it within their products. Google’s ChromeOS and Android have roots in Linux. Many of the corporations that make use of Linux provide fixes and new features for Linux as they use the software for their businesses. These improvements are given back to the Linux community and Linux improves as a result. These efforts on the part of the developer community is how we can continually improve and grow without having to charge our users money.

Linux Advantage: Community

Whether you are a home user of Linux, a Linux software or application developer, or an employee of an organization that uses the operating system, you are a member of the Linux and Open Source communities and you benefit from the efforts of the developers who contribute to Linux. Members of the Linux community can – and do – run

Linux on almost any hardware, from the prettiest Macbook to the cheapest netbook, from the newest Chromebook to some very old machines designed for Windows, and from the most powerful Internet servers to the smallest smart thermostat.

Having “an inspiring, engaging, and enjoyable community” (Preface: The Art of Community, 2nd Edition) is the lifeblood of any open source software project. The community provides product and feature ideas, user support, developer talent, documentation, financial support, visionary direction, and cultural norms – for the benefit of anyone who uses, contributes to, or otherwise supports the project. Although many projects, applications and even companies have their own communities, the inspirational engagement of the Linux community is one of the key things that makes Linux one of the top 3 operating systems in the world.

Linux Security

The Linux operating system is more secure, and better supported than the operating systems preinstalled on most home computer hardware today. Linux is backed by many large corporations, as well as independent developers and users, many of whom are focused on ensuring and improving the security that is built into the operating system. The built-in updater provided with your Linux distribution provides security updates for both its software applications and the operating system. Vulnerabilities are patched more quickly, and are delivered automatically and more frequently than the two most popular operating systems.

Combination lock

Four Reasons Why Linux Is More Secure

When you use a distribution of Linux, security updates, driver updates, application updates, software upgrades and operating system upgrades are all provided, all free of charge. And they are all available from trusted sources. So you have no more need to search the Internet for software. No more risking malware or junkware infections as a result from downloading from the wrong site. There are thousands of software titles in hundreds of categories available in your Linux distribution’s repositories – the ultimate in a trusted source!

Linux is designed with security in mind. Unlike operating systems that update only once a month, Linux distributions receive updates continuously. The updates include security patches for the operating system and all of its components. Security updates for all of its installed applications are also provided on the same schedule. This ensures that you

have the latest protection for all of your computer's software – as soon as it's available!

Linux can get viruses and other infections... but, as a rule, it doesn't. Rapid and timely updates ensure that there are very few, if any threats to Linux systems that persist in the wild. In reality, there have been very few «public» infections in the last 10 years that can affect even Linux. And because of security updates to Linux, those few old attacks are no longer a threat to anyone installing or using a modern Linux distribution today. Linux is designed to make it difficult for viruses, rootkits and other malware to be installed and run without conscious intervention by you, the user. Even if you do accidentally invite in an infection, chances are it's designed to attack Windows and can do no damage to your Linux system.

Another significant security feature of Linux is that its users are not administrators by default. Administrators («root» users) on any computer system have permission to do anything they want, including do damage to the system. For example, other operating systems look at the name of a file to determine which program should open it, then immediately attempt to open it. That design makes it easy for an intruder to attack a computer. Linux opens a file based on what the file is, not based on its name. So even if a malicious program disguises its identity by using a name like «Business Proposal.docx» Linux will recognize the file as a program. The system provides a warning that the file is not a text document, but that it is really a program that will be run if you give it permission to continue. To be extra secure, Linux requires you to provide your administrator password to grant that permission. Every single time.

Conclusion

Unlike Windows, and OSX, Linux is not created and supported by just one company. Over 4,000 individual developers contributed to Linux over the last 15 years. Linux is supported by individuals, the Linux and Open Source communities, as well as many organizations. These organizations include Intel, Redhat, Linaro, Samsung, IBM, SUSE, Texas Instruments, Google, Canonical, Oracle, AMD, and Microsoft. These corporations, and others, use the Linux operating system to run their businesses, or include it within their products. (Google Android phones and Chromebooks, Samsung televisions, etc.) They want to ensure that Linux is provided with the best protection from security vulnerabilities. Many of these corporations provide security fixes and new security measures for Linux as they use it in their businesses. These improvements are given back to the Linux distribution and the software improves. Whether you are a home

user of Linux, a Linux software or application developer, or an employee of a company that uses Linux, the scrutiny and ongoing security improvements provided for Linux are benefiting you.

(6,483 symbols)

http://goinglinux.com/articles/CommunityTheLinuxAdvantage_en.htm
http://goinglinux.com/articles/Security-TheLinuxAdvantage_en.htm

TEXT 5 COMPUTERS: HISTORY AND DEVELOPMENT

Nothing epitomizes modern life better than the computer. For better or worse, computers have infiltrated every aspect of our society. Today computers do much more than simply compute: supermarket scanners calculate our grocery bill while keeping store inventory; computerized telephone switching centers play traffic cop to millions of calls and keep lines of communication untangled; and automatic teller machines let us conduct banking transactions from virtually anywhere in the world. But where did all this technology come from and where is it heading? To fully understand and appreciate the impact computers have on our lives and promises they hold for the future, it is important to understand their evolution.

Early Computing Machines and Inventors

The abacus, which emerged about 5,000 years ago in Asia Minor and is still in use today, may be considered the first computer. This device allows users to make computations using a system of sliding beads arranged on a rack. Early merchants used the abacus to keep trading transactions. But as the use of paper and pencil spread, particularly in Europe, the abacus lost its importance. It took nearly 12 centuries, however, for the next significant advance in computing devices to emerge. In 1642, Blaise Pascal (1623–1662), the 18-year-old son of a French tax collector, invented what he called a numerical wheel calculator to help his father with his duties. This brass rectangular box, also called a Pascaline, used eight movable dials to add sums up to eight figures long. Pascal's device used a base of ten to accomplish this. For example, as one dial moved ten notches, or one complete revolution, it moved the next dial – which represented the ten's column – one place. When the ten's dial moved one revolution, the dial representing the hundred's place moved one notch and so on. The drawback to the Pascaline, of course, was its limitation to addition.

In 1694, a German mathematician and philosopher, Gottfried Wilhem von Leibniz (1646–1716), improved the Pascaline by creating a machine that could also multiply. Like its predecessor, Leibniz's mechanical multiplier worked by a system of gears and dials. Partly by studying Pascal's original notes and drawings, Leibniz was able to refine his machine. The centerpiece of the machine was its stepped-drum gear design, which offered an elongated version of the simple flat gear. It wasn't until 1820, however, that mechanical calculators gained widespread use. Charles Xavier Thomas de Colmar, a Frenchman, invented a machine that could perform the four basic arithmetic functions. Colmar's mechanical calculator, the arithometer, presented a more practical approach to computing because it could add, subtract, multiply and divide. With its enhanced versatility, the arithometer was widely used up until the First World War. Although later inventors refined Colmar's calculator, together with fellow inventors Pascal and Leibniz, he helped define the age of mechanical computation.

The real beginnings of computers as we know them today, however, lay with an English mathematics professor, Charles Babbage (1791–1871). Frustrated at the many errors he found while examining calculations for the Royal Astronomical Society, Babbage declared, “I wish to God these calculations had been performed by steam!” With those words, the automation of computers had begun. By 1812, Babbage noticed a natural harmony between machines and mathematics: machines were best at performing tasks repeatedly without mistake; while mathematics, particularly the production of mathematic tables, often required the simple repetition of steps. The problem centered on applying the ability of machines to the needs of mathematics. Babbage's first attempt at solving this problem was in 1822 when he proposed a machine to perform differential equations, called a Difference Engine. Powered by steam and large as a locomotive, the machine would have a stored program and could perform calculations and print the results automatically. After working on the Difference Engine for 10 years, Babbage was suddenly inspired to begin work on the first general-purpose computer, which he called the Analytical Engine. Babbage's assistant, Augusta Ada King, Countess of Lovelace (1815–1842) and daughter of English poet Lord Byron, was instrumental in the machine's design. One of the few people who understood the Engine's design as well as Babbage, she helped revise plans, secure funding from the British government, and communicate the specifics of the Analytical Engine to the public. Also, Lady Lovelace's fine

understanding of the machine allowed her to create the instruction routines to be fed into the computer, making her the first female computer programmer. In the 1980's, the U.S. Defense Department named a programming language ADA in her honor.

Babbage's steam-powered Engine, although ultimately never constructed, may seem primitive by today's standards. However, it outlined the basic elements of a modern general purpose computer and was a breakthrough concept. Consisting of over 50,000 components, the basic design of the Analytical Engine included input devices in the form of perforated cards containing operating instructions and a “store” for memory of 1,000 numbers of up to 50 decimal digits long. It also contained a “mill” with a control unit that allowed processing instructions in any sequence, and output devices to produce printed results. Babbage borrowed the idea of punch cards to encode the machine's instructions from the Jacquard loom. The loom, produced in 1820 and named after its inventor, Joseph-Marie Jacquard, used punched boards that controlled the patterns to be woven.

In 1889, an American inventor, Herman Hollerith (1860–1929), also applied the Jacquard loom concept to computing. His first task was to find a faster way to compute the U.S. census. The previous census in 1880 had taken nearly seven years to count and with an expanding population, the bureau feared it would take 10 years to count the latest census. Unlike Babbage's idea of using perforated cards to instruct the machine, Hollerith's method used cards to store data information which he fed into a machine that compiled the results mechanically. Each punch on a card represented one number, and combinations of two punches represented one letter. As many as 80 variables could be stored on a single card. Instead of ten years, census takers compiled their results in just six weeks with Hollerith's machine. In addition to their speed, the punch cards served as a storage method for data and they helped reduce computational errors. Hollerith brought his punch card reader into the business world, founding Tabulating Machine Company in 1896, later to become International Business Machines (IBM) in 1924 after a series of mergers. Other companies such as Remington Rand and Burroughs also manufactured punch readers for business use. Both business and government used punch cards for data processing until the 1960's.

In the ensuing years, several engineers made other significant advances. Vannevar Bush (1890–1974) developed a calculator for solving differential equations in 1931. The machine could solve complex

differential equations that had long left scientists and mathematicians baffled. The machine was cumbersome because hundreds of gears and shafts were required to represent numbers and their various relationships to each other. To eliminate this bulkiness, John V. Atanasoff (b. 1903), a professor at Iowa State College (now called Iowa State University) and his graduate student, Clifford Berry, envisioned an all-electronic computer that applied Boolean algebra to computer circuitry. This approach was based on the mid-19th century work of George Boole (1815–1864) who clarified the binary system of algebra, which stated that any mathematical equations could be stated simply as either true or false. By extending this concept to electronic circuits in the form of on or off, Atanasoff and Berry had developed the first all-electronic computer by 1940. Their project, however, lost its funding and their work was overshadowed by similar developments by other scientists.

(6,591 symbols)

http://www.dia.eui.upm.es/asignatu/sis_op1/comp_hd/comp_hd.htm

TEXT 5.1 FIVE GENERATIONS OF MODERN COMPUTERS

First Generation (1945–1956)

With the onset of the Second World War, governments sought to develop computers to exploit their potential strategic importance. This increased funding for computer development projects hastened technical progress. By 1941 German engineer Konrad Zuse had developed a computer, the Z3, to design airplanes and missiles. The Allied forces, however, made greater strides in developing powerful computers. In 1943, the British completed a secret code-breaking computer called Colossus to decode German messages. The Colossus's impact on the development of the computer industry was rather limited for two important reasons. First, Colossus was not a general-purpose computer; it was only designed to decode secret messages. Second, the existence of the machine was kept secret until decades after the war.

American efforts produced a broader achievement. Howard H. Aiken (1900–1973), a Harvard engineer working with IBM, succeeded in producing an all-electronic calculator by 1944. The purpose of the computer was to create ballistic charts for the U.S. Navy. It was about half as long as a football field and contained about 500 miles of wiring. The Harvard-IBM Automatic Sequence Controlled Calculator, or Mark I for short, was a electronic relay computer. It used electromagnetic signals

to move mechanical parts. The machine was slow (taking 3–5 seconds per calculation) and inflexible (in that sequences of calculations could not change); but it could perform basic arithmetic as well as more complex equations.

Another computer development spurred by the war was the Electronic Numerical Integrator and Computer (ENIAC), produced by a partnership between the U.S. government and the University of Pennsylvania. Consisting of 18,000 vacuum tubes, 70,000 resistors and 5 million soldered joints, the computer was such a massive piece of machinery that it consumed 160 kilowatts of electrical power, enough energy to dim the lights in an entire section of Philadelphia. Developed by John Presper Eckert (1919–1995) and John W. Mauchly (1907–1980), ENIAC, unlike the Colossus and Mark I, was a general-purpose computer that computed at speeds 1,000 times faster than Mark I.

In the mid-1940's John von Neumann (1903–1957) joined the University of Pennsylvania team, initiating concepts in computer design that remained central to computer engineering for the next 40 years. Von Neumann designed the Electronic Discrete Variable Automatic Computer (EDVAC) in 1945 with a memory to hold both a stored program as well as data. This «stored memory» technique as well as the «conditional control transfer», that allowed the computer to be stopped at any point and then resumed, allowed for greater versatility in computer programming. The key element to the von Neumann architecture was the central processing unit, which allowed all computer functions to be coordinated through a single source. In 1951, the UNIVAC I (Universal Automatic Computer), built by Remington Rand, became one of the first commercially available computers to take advantage of these advances. Both the U.S. Census Bureau and General Electric owned UNIVACs. One of UNIVAC's impressive early achievements was predicting the winner of the 1952 presidential election, Dwight D. Eisenhower.

First generation computers were characterized by the fact that operating instructions were made-to-order for the specific task for which the computer was to be used. Each computer had a different binary-coded program called a machine language that told it how to operate. This made the computer difficult to program and limited its versatility and speed. Other distinctive features of first generation computers were the use of vacuum tubes (responsible for their breathtaking size) and magnetic drums for data storage.

Second Generation Computers (1956–1963)

By 1948, the invention of the transistor greatly changedthe computer's development. The transistor replaced the large, cumbersome vacuum tube in televisions, radios and computers. As a result, the size of electronic machinery has been shrinking ever since. The transistor was at work in the computer by 1956. Coupled with early advances in magnetic-core memory, transistors led to second generation computers that were smaller, faster, more reliable and more energy-efficient than their predecessors. The first large-scale machines to take advantage of this transistor technology were early supercomputers, Stretch by IBM and LARC by Sperry-Rand. These computers, both developed for atomic energy laboratories, could handle an enormous amount of data, a capability much in demand by atomic scientists. The machines were costly, however, and tended to be too powerful for the business sector's computing needs, thereby limiting their attractiveness. Only two LARCs were ever installed: one in the Lawrence Radiation Labs in Livermore, California, for which the computer was named (Livermore Atomic Research Computer) and the other at the U.S. Navy Research and Development Center in Washington, D.C. Second generation computers replaced machine language with assembly language, allowing abbreviated programming codes to replace long, difficult binary codes.

Throughout the early 1960's, there were a number of commercially successful second generation computers used in business, universities, and government from companies such as Burroughs,Control Data, Honeywell, IBM, Sperry-Rand, and others. These second generation computers were also of solid state design, and contained transistors in place of vacuum tubes. They also contained all the components we associate with the modern day computer: printers, tape storage, disk storage, memory, operating systems, and stored programs. One important example was the IBM 1401, which was universally accepted throughout industry, and is considered by many to be the Model T of the computer industry. By 1965, most large business routinely processed financial information using second generation computers.

It was the stored program and programming language that gave computers the flexibility to finally be cost effective and productive for business use. The stored program concept meant that instructions to run a computer for a specific function (known as a program) were held inside the computer's memory, and could quickly be replaced by a different set of instructions for a different function. A computer could print customer

invoices and minutes later design products or calculate paychecks. More sophisticated high-level languages such as COBOL (Common Business-Oriented Language) and FORTRAN (Formula Translator) came into common use during this time, and have expanded to the current day. These languages replaced cryptic binary machine code with words, sentences, and mathematical formulas, making it much easier to program a computer. New types of careers (programmer, analyst, and computer systems expert) and the entire software industry began with second generation computers.

Third Generation Computers (1964–1971)

Though transistors were clearly an improvement over the vacuum tube, they still generated a great deal of heat, which damaged the computer's sensitive internal parts. The quartz rock eliminated this problem. Jack Kilby, an engineer with Texas Instruments, developed the integrated circuit (IC) in 1958. The IC combined three electronic components onto a small silicon disc, which was made from quartz. Scientists later managed to fit even more components on a single chip, called a semiconductor. As a result, computers became ever smaller as more components were squeezed onto the chip. Another third-generation development included the use of an operating system that allowed machines to run many different programs at once with a central program that monitored and coordinated the computer's memory.

Fourth Generation (1971–Present)

After the integrated circuits, the only place to go was down – in size, that is. Large scale integration (LSI) could fit hundreds of components onto one chip. By the 1980's, very large scale integration (VLSI) squeezed hundreds of thousands of components onto a chip. Ultra-large scale integration (ULSI) increased that number into the millions. The ability to fit so much onto an area about half the size of a U.S. dime helped diminish the size and price of computers. It also increased their power, efficiency and reliability. The Intel 4004 chip, developed in 1971, took the integrated circuit one step further by locating all the components of a computer (central processing unit, memory, and input and output controls) on a minuscule chip. Whereas previously the integrated circuit had had to be manufactured to fit a special purpose, now one microprocessor could be manufactured and then programmed to meet any number of demands. Soon everyday household items such as microwave ovens, television sets and automobiles with electronic fuel injection incorporated microprocessors.

Such condensed power allowed everyday people to harness a computer's power. They were no longer developed exclusively for large

business or government contracts. By the mid-1970's, computer manufacturers sought to bring computers to general consumers. These minicomputers came complete with user-friendly software packages that offered even non-technical users an array of applications, most popularly word processing and spreadsheet programs. Pioneers in this field were Commodore, Radio Shack and Apple Computers. In the early 1980's, arcade video games such as Pac Man and home video game systems such as the Atari 2600 ignited consumer interest for more sophisticated, programmable home computers.

In 1981, IBM introduced its personal computer (PC) for use in the home, office and schools. The 1980's saw an expansion in computer use in all three arenas as clones of the IBM PC made the personal computer even more affordable. The number of personal computers in use more than doubled from 2 million in 1981 to 5.5 million in 1982. Ten years later, 65 million PCs were being used. Computers continued their trend toward a smaller size, working their way down from desktop to laptop computers (which could fit inside a briefcase) to palmtop (able to fit inside a breast pocket). In direct competition with IBM's PC was Apple's Macintosh line, introduced in 1984. Notable for its user-friendly design, the Macintosh offered an operating system that allowed users to move screen icons instead of typing instructions. Users controlled the screen cursor using a mouse, a device that mimicked the movement of one's hand on the computer screen.

As computers became more widespread in the workplace, new ways to harness their potential developed. As smaller computers became more powerful, they could be linked together, or networked, to share memory space, software, information and communicate with each other. As opposed to a mainframe computer, which was one powerful computer that shared time with many terminals for many applications, networked computers allowed individual computers to form electronic co-ops. Using either direct wiring, called a Local Area Network (LAN), or telephone lines, these networks could reach enormous proportions. A global web of computer circuitry, the Internet, for example, links computers worldwide into a single network of information. During the 1992 U.S. presidential election, vice-presidential candidate Al Gore promised to make the development of this so-called “information superhighway” an administrative priority. Though the possibilities envisioned by Gore and others for such a large network are often years (if not decades) away from realization, the most popular use today for computer networks such as the

Internet is electronic mail, or E-mail, which allows users to type in a computer address and send messages through networked terminals across the office or across the world.

Fifth Generation (Present and Beyond)

Defining the fifth generation of computers is somewhat difficult because the field is in its infancy. The most famous example of a fifth generation computer is the fictional HAL9000 from Arthur C. Clarke's novel, 2001: A Space Odyssey. HAL performed all of the functions currently envisioned for real-life fifth generation computers. With artificial intelligence, HAL could reason well enough to hold conversations with its human operators, use visual input, and learn from its own experiences. (Unfortunately, HAL was a little too human and had a psychotic breakdown, commandeering a spaceship and killing most humans on board).

Though the wayward HAL9000 may be far from the reach of real-life computer designers, many of its functions are not. Using recent engineering advances, computers may be able to accept spoken word instructions and imitate human reasoning. The ability to translate a foreign language is also a major goal of fifth generation computers. This feat seemed a simple objective at first, but appeared much more difficult when programmers realized that human understanding relies as much on context and meaning as it does on the simple translation of words.

Many advances in the science of computer design and technology are coming together to enable the creation of fifth-generation computers. Two such engineering advances are parallel processing, which replaces von Neumann's single central processing unit design with a system harnessing the power of many CPUs to work as one. Another advance is superconductor technology, which allows the flow of electricity with little or no resistance, greatly improving the speed of information flow. Computers today have some attributes of fifth generation computers. For example, expert systems assist doctors in making diagnoses by applying the problem-solving steps a doctor might use in assessing a patient's needs. It will take several more years of development before expert systems are in widespread use.

(11,237 symbols)

http://www.dia.eui.upm.es/asignatu/sis_op1/comp_hd/comp_hd.htm

TEXT 6 HOW COMPUTER MOUSE WAS INVENTED

The trackball, a related pointing device, was invented in 1941 by Ralph Benjamin as part of a World War II-era fire-control radar plotting system called Comprehensive Display System (CDS). Benjamin was then working for the British Royal Navy Scientific Service. Benjamin's project used analog computers to calculate the future position of target aircraft based on several initial input points provided by a user with a joystick. Benjamin felt that a more elegant input device was needed and invented what they called a "roller ball" for this purpose.

The device was patented in 1947, but only a prototype using a metal ball rolling on two rubber-coated wheels was ever built, and the device was kept as a military secret.

Another early trackball was built by British electrical engineer Kenyon Taylor in collaboration with Tom Cranston and Fred Longstaff. Taylor was part of the original Ferranti Canada, working on the Royal Canadian Navy's DATAR (Digital Automated Tracking and Resolving) system in 1952.

DATAR was similar in concept to Benjamin's display. The trackball used four disks to pick up motion, two each for the X and Y directions. Several rollers provided mechanical support. When the ball was rolled, the pickup discs spun and contacts on their outer rim made periodic contact with wires, producing pulses of output with each movement of the ball. By counting the pulses, the physical movement of the ball could be determined. A digital computer calculated the tracks, and sent the resulting data to other ships in a task force using pulse-code modulation radio signals. This trackball used a standard Canadian five-pin bowling ball. It was not patented, as it was a secret military project as well.

On 2 October 1968, a mouse device named *Rollkugel* (German for "rolling ball") was released that had been developed and published by the German company Telefunken. As the name suggests and unlike Engelbart's mouse, the Telefunken model already had a ball. It was based on an earlier trackball-like device (also named *Rollkugel*) that was embedded into radar flight control desks. This had been developed around 1965 by a team led by Rainer Mallebrein at Telefunken Konstanz for the German *Bundesanstalt für Flugsicherung* as part of their TR 86 process computer system with its SIG 100-86 vector graphics terminal.

When the development for the Telefunken main frame TR 440 (de) began in 1965, Mallebrein and his team came up with the idea of «reversing» the existing Rollkugel into a moveable mouse-like device,

so that customers did not have to be bothered with mounting holes for the earlier trackball device. Together with light pens and trackballs, it was offered as optional input device for their system since 1968. Some samples, installed at the Leibniz-Rechenzentrum in Munich in 1972, are still well preserved. Telefunken considered the invention too small to apply for a patent on their device.

A few months after Telefunken started to sell the Rollkugel, Engelbart released his demo on 9 December 1968. Independently, Douglas Engelbart at the Stanford Research Institute (now SRI International) invented his first mouse prototype in the 1960s with the assistance of his lead engineer Bill English. They christened the device the *mouse* as early models had a cord attached to the rear part of the device looking like a tail and generally resembling the common mouse. Engelbart never received any royalties for it, as his employer SRI held the patent, which ran out before it became widely used in personal computers. The invention of the mouse was just a small part of Engelbart's much larger project, aimed at augmenting human intellect via the Augmentation Research Center.

Several other experimental pointing-devices developed for Engelbart's oN-Line System (NLS) exploited different body movements – for example, head-mounted devices attached to the chin or nose – but ultimately the mouse won out because of its speed and convenience. The first mouse, a bulky device used two potentiometers perpendicular to each other and connected to wheels: the rotation of each wheel translated into motion along one axis. At the time of the «Mother of All Demos», Englebart's group had been using their second generation, 3-button mouse for about a year.

The Xerox Alto was one of the first computers designed for individual use in 1973, and is regarded as the grandfather of computers that utilize the mouse. Inspired by PARC's Alto, the Lilith, a computer which had been developed by a team around Niklaus Wirth at ETH Zürich between 1978 and 1980, provided a mouse as well. The third marketed version of an integrated mouse shipped as a part of a computer and intended for personal computer navigation came with the Xerox 8010 Star Information System in 1981.

By 1982 the Xerox 8010 was probably the best-known computer with a mouse, and the forthcoming Apple Lisa was rumored to use one, but the peripheral remained obscure; Jack Hawley of The Mouse House reported that one buyer for a large organization believed at first that his company sold lab mice. Hawley, who manufactured mice for Xerox, stated that

“Practically, I have the market all to myself right now”; a Hawley mouse cost $415.

That year Microsoft made the decision to make the MS-DOS program Microsoft Word mouse-compatible, and developed the first PC-compatible mouse. Microsoft's mouse shipped in 1983, thus beginning Microsoft hardware. However, the mouse remained relatively obscure until the 1984 appearance of the Macintosh 128K, which included an updated version of the Lisa Mouse and the Atari ST in 1985.

(4,454 symbols)

https://www.quora.com/How-was-computer-mouse-invented

TEXT 7 WEB BROWSERS

A web browser is a special software program (application) used to retrieve files from remote web servers. A web browser can open Web-based HTML files, FTP connections, graphic images and other files. The browser application is smart enough to be able to tell the difference between these files and display them properly. Browsers are also created to be “intelligent” enough to be able to “learn” to handle even more types of files using “plug-ins”.

Web browsers are software. They run on your computer and do not connect you to the Internet. You use a web browser after you connect to your Internet Provider. A browser is not an online service like America Online, MSN or Compuserve. The online service provider provides telephone numbers and dial up connections. A web browser uses that connection to reach across the Internet and download files and information.

Now, you should know that America Online purchased the organization that produced the Netscape browser. Because there was great confusion about what the Internet and Internet Service Providers are, the online service «Netscape» was created to take advantage of the confusion between web browsers and the Internet.

Netscape and Microsoft Internet Explorer are applications, not Internet Service Providers. There now exists a «Netscape» Internet Service Provider.

Browser Applications

The most well known browsers are listed below. The order of the listing of the browsers is in relation to the number of copies of the software installed on computers. Microsoft Internet Explorer tops the list only because it is automatically installed with the operating system and there are no options to remove MSIE from the installation in versions of Windows

prior to Windows XP. By U.S. Government court order Microsoft has added a feature to Windows that will allow you to uninstall most of MSIE's functionality; however, Microsoft completely integrated the browser into the operating system and completely removing all functions of the browser would actually damage the operating system, or so was their argument to the court.

Microsoft Internet Explorer

This browser is automatically installed with Windows. No, you don't have a choice about it. You CAN set another browser as your default however.

Netscape Navigator / Communicator

A free download. AOL/Time-Warner owns Netscape now and appears actively engaged in driving this browser into the ground.

Mozilla

A (better) freeware variation of Netscape. Uses the Gecko engine originally developed for Mosaic and the later Netscape spin-off. Mozilla itself is an open source spin-off of the original code that was used to create Netscape. Mozilla's biggest feature is that it uses tabs for various web pages, saving desktop space. Developed using open source.

Opera (not free) – Originally a stripped-down browser offering less functionality but greater page rendering speed.

NCSA Mosaic – The FIRST web browser. No longer under active development. Last version 3.0 supports WinNT and 2.1.1 for MacIntosh.

Hot Java – From Sun Microsystems. No longer under development since Solaris 8 and later are shipped with Netscape.

Lynx – Text Only – No graphics, tables or frames, but runs from DOS.

Browser Functions and Components

Render Engine

Every web browser has an HTML rendering engine designed to read the embedded HTML tags and use those tags to arrange content and format the text on the web page. The rendering engine is the guts of the web browser and no two browsers will render a web page the same way. This is why any good web developer will test the web page in several web browsers (Internet Explorer, Netscape, Mozilla) that run on several different platforms (Windows, Linux, Unix and Mac).

Cache

When a web browser downloads a web page, it stores that web page in a special location on the computer called the cache. Storing web pages and

the content inside them allows the browser to skip re-downloading that content if the web page have not been changed on the web server. This speeds up the web browsing experience, particularly when hitting the back button to return to the previous web page. The cache contains all the files you have pulled and viewed in your web browser. In most cases, web pages and files are left on your computer until a certain size limit is reached. At that point, the browser will delete older files from previous web sessions and replace them with newer files from whatever web server you are browsing.

If your caching functions are set to the defaults, you will have difficulty with dynamically generated pages. Sites such as CNN and CNBC change the content of their web pages several times a day. The default settings in the web browser will result in your seeing the first web page you browse to for as long as the browser is open. You can change these settings so that in just a minute past when the content changed at the site, your web browser will pull the new web page.

Bookmarks and Shortcuts

To help you find things again later, most browsers offer a bookmark function to allow you to store the address of the page in a list of favorite sites. This list of favorites can be browsed later and when the listed site is clicked on, the site's web page is retrieved from the server. Netscape calls these 'bookmarks' and stores them in a bookmarks folder. Internet Explorer calls them 'shortcuts' and stores them in a Favorites list.

Plug-Ins

Plug-ins are software you can download and install on your computer that extends the functionality of your web browser. Web browsers were designed to support this functionality to allow web browsers to become smarter over time. This plug-in functionality was used to provide additional capabilities.

Plugins run 'applets' or handle different kinds of media files, such as audio (files with names ending in .ogg, .wav, .mp3, .rm, .ram and more) and video (files ending in .avi, .mpeg, .mov and .qt). The Macromedia Flash plugin runs Flash applets (.swf). Sun's Java plugin runs Java 'applets'

(4,838 symbols)

http://www.inetdaemon.com/tutorials/www/web_browser/

TEXT 8 WHAT IS A NETWORK?

A network is set of computers linked together for the purpose of communicating and sharing information. The Internet is a global super-network, so is the local area network (a LAN) at your workplace or your school, as is the wireless hotspot at your local coffee shop, hotel or library, the telephone and cellular systems, and the satellite communications in space.

What defines a network is often defined by who owns and operates the equipment and the computers that are part of the network. Thus, your school's network is separate from the Internet.

You know you have a network when you have two or more computers connected together and they are able to communicate. Plugged into the back of each computer is some sort of communications port. Nearly all computers today have one or more serial ports, parallel ports, Ethernet ports, modem ports, fire wire ports, USB ports and more. All of these ports can be used in one way or another to connect computers to a network. The most common type of network port is an Ethernet port (the square port with the row of connectors on the bottom). The next most common is a wireless network connection, but that has no physical connector port.

Xerox was the first company to research and develop a network. Once upon a time, Xerox printers were extremely expensive, so companies wanted to share them. Xerox knew their printers were expensive and users were only able to print from one big computer (a mainframe) attached to the printer directly. You would print your document, and then walk down to the building next door where the mainframe was housed, with the printer, and pick up your printout. Xerox decided that they could sell more printers if they could make it possible for anyone to use the printer from any computer. To allow multiple computers to communicate with the printer, some means of sharing a connection to the printer was needed. Xerox put Bob Metcalf and others to work on researching and designing what eventually came to be called Ethernet. Ethernet is now the most common networking protocol on the planet.

Hosts, End Stations and Workstations

When people talk about networks, they often refer to computers that are at the edge of the network as hosts, end stations, workstations, or servers. Its all just the same thing, a computer attached to the network; though the word HOST has the most general meaning and can include

anything attached to the network including hubs, bridges, switches, routers, access points, firewalls, workstations, servers, mainframes, printers, scanners, copiers, fax machines and more!

Just about everything electronic that has a processor and which you would use in an office is 'network capable' today and lots of things that aren't currently networked probably will be networked in the future. In many offices the phone system already IS the network (Voice over IP).

Hosts, End Stations and Workstations

When people talk about networks, they often refer to computers that are at the edge of the network as hosts, end stations, workstations, or servers. Its all just the same thing, a computer attached to the network; though the word HOST has the most general meaning and can include anything attached to the network including hubs, bridges, switches, routers, access points, firewalls, workstations, servers, mainframes, printers, scanners, copiers, fax machines and more!

Just about everything electronic that has a processor and which you would use in an office is 'network capable' today and lots of things that aren't currently networked probably will be networked in the future. In many offices the phone system already IS the network (Voice over IP).

LAN, MAN, WAN and er.. IPAN??

There are some terms, acronyms actually, that are used to describe the size and scope of a network: LAN, WAN, MAN. We've added our own term 'IPAN'

LAN

A Local Area Network (LAN) is usually a single set of connected computers that are in a single small location such as a room, a floor of a building, or the whole building.

MAN

A Metropolitan Area Network (MAN) is a network that encompasses a city or town. It is usually multiple point-to-point fiber-optic connections put together by a communications company and leased to their customers, but a small number of big corporations have built a few of these of their own and opened them to the local companies with which they do business. The automotive, travel and insurance industries are just a few examples of who has built a WAN.

WAN

A Wide Area Network (WAN) is usually composed of all the links that connect the buildings of a campus together, such as at a University or at a corporate headquarters. WAN connections can often span miles, so

you frequently hear people referring to the 'WAN' connection to an office half way around the world. Usually, what distinguishes a WAN from a LAN is that there are one or more links that span a large distance over serial, T-carrier or ISDN, Frame Relay or ATM links.

IPAN

So what the heck is an IPAN? An IPAN is an Inter-Planetary Area Network. NASA has built a Deep Space Internet that uses a store-and-forward communications protocol called Disruption Tolerant Networking (DTN). The mechanical rovers Spirit and Opportunity on the planet Mars, were given addresses on a NASA network and NASA uses Internet and IPAN protocols to communicate with the Mars rovers. While the communication with the rovers never crosses over the Internet, the NASA network does have hosts spanning space between the planets Earth and Mars. They also have probes they have sent into the outer solar system with which they use IPAN to communicate.

This networking tutorial section will teach you about running an Internet Protocol (IP)-based network on top of Ethernet. A typical physically wired network is built with several layers of technologies layered one top of one another. The TCP-Model and the OSI Model tutorials will help you understand the layering concepts and you should probably start there first and come back to this page.

This list starts with the lower layer functions or protocols and works its way up.

All networks have several layers of functions stacked on top of each other. Ethernet is used to provide the means to transmit information encoded in electrical signals across copper wiring between two computers. Internet Protocol networking software running on the computers use the Ethernet network to send data back and forth inside IP packets. The Internet Protocol layer provides the means for the computer to connect to the network, obtain a logical address, to learn the logical addresses of other computers and to communicate with the other computers on the network. Internet Protocol provides the basic network functions.

(5,437 symbols)

http://www.inetdaemon.com/tutorials/networking/

TEXT 9 WHAT IS AN IP ADDRESS?

Every machine on a network has a unique identifier. Just as you would address a letter to send in the mail, computers use the unique identifier to send data to specific computers on a network. Most networks today,

including all computers on the Internet, use the TCP/IP protocol as the standard for how to communicate on the network. In the TCP/IP protocol, the unique identifier for a computer is called its IP address.

There are two standards for IP addresses: IP Version 4 (IPv4) and IP Version 6 (IPv6). All computers with IP addresses have an IPv4 address, and many are starting to use the new IPv6 address system as well. Here's what these two address types mean:

- IPv4 uses 32 binary bits to create a single unique address on the network. An IPv4 address is expressed by four numbers separated by dots. Each number is the decimal (base-10) representation for an eight-digit binary (base-2) number, also called an octet. For example: 216.27.61.137
- IPv6 uses 128 binary bits to create a single unique address on the network. An IPv6 address is expressed by eight groups of hexadecimal (base-16) numbers separated by colons, as in 2001:cdba:0000:0000:0000:0000:3257:9652. Groups of numbers that contain all zeros are often omitted to save space, leaving a colon separator to mark the gap (as in 2001:cdba::3257:9652).

At the dawn of IPv4 addressing, the Internet was not the large commercial sensation it is today, and most networks were private and closed off from other networks around the world. When the Internet exploded, having only 32 bits to identify a unique Internet address caused people to panic that we'd run out of IP addresses. Under IPv4, there are 232 possible combinations, which offers just under 4.3 billion unique addresses. IPv6 raised that to a panic-relieving 2128 possible addresses. Later, we'll take a closer look at how to understand your computer's IPv4 or IPv6 addresses.

How does your computer get its IP address? An IP address can be either dynamic or static. A static address is one that you configure yourself by editing your computer's network settings. This type of address is rare, and it can create network issues if you use it without a good understanding of TCP/IP. Dynamic addresses are the most common. They're assigned by the Dynamic Host Configuration Protocol (DHCP), a service running on the network. DHCP typically runs on network hardware such as routers or dedicated DHCP servers.

Dynamic IP addresses are issued using a leasing system, meaning that the IP address is only active for a limited time. If the lease expires, the computer will automatically request a new lease. Sometimes, this means the computer will get a new IP address, too, especially if the computer was unplugged from the network between leases. This process is usually

transparent to the user unless the computer warns about an IP address conflict on the network (two computers with the same IP address). An address conflict is rare, and today's technology typically fixes the problem automatically.

Next, let's take a closer look at the important parts of an IP address and the special roles of certain addresses.

IP Classes PREV NEXT

Earlier, you read that IPv4 addresses represent four eight-digit binary numbers. That means that each number could be 00000000 to 11111111 in binary, or 0 to 255 in decimal (base-10). In other words, 0.0.0.0 to 255.255.255.255. However, some numbers in that range are reserved for specific purposes on TCP/IP networks. These reservations are recognized by the authority on TCP/IP addressing, the Internet Assigned Numbers Authority (IANA). Four specific reservations include the following:

- **0.0.0.0** – This represents the default network, which is the abstract concept of just being connected to a TCP/IP network.
- **255.255.255.255** – This address is reserved for network broadcasts, or messages that should go to all computers on the network.
- **127.0.0.1** – This is called the loopback address, meaning your computer's way of identifying itself, whether or not it has an assigned IP address.
- **169.254.0.1 to 169.254.255.254** – This is the Automatic Private IP Addressing (APIPA) range of addresses assigned automatically when a computer's unsuccessful getting an address from a DHCP server.

The other IP address reservations are for subnet classes. A subnet is a smaller network of computers connected to a larger network through a router. The subnet can have its own address system so computers on the same subnet can communicate quickly without sending data across the larger network. A router on a TCP/IP network, including the Internet, is configured to recognize one or more subnets and route network traffic appropriately. The following are the IP addresses reserved for subnets:

- 10.0.0.0 to 10.255.255.255 – This falls within the Class A address range of 1.0.0.0 to 127.0.0.0, in which the first bit is 0.
- 172.16.0.0 to 172.31.255.255 – This falls within the Class B address range of 128.0.0.0 to 191.255.0.0, in which the first two bits are 10.
- 192.168.0.0 to 192.168.255.255 – This falls within the Class C range of 192.0.0.0 through 223.255.255.0, in which the first three bits are 110.

- Multicast (formerly called Class D) – The first four bits in the address are 1110, with addresses ranging from 224.0.0.0 to 239.255.255.255.
- Reserved for future/experimental use (formerly called Class E) – addresses 240.0.0.0 to 254.255.255.254.

The first three (within Classes A, B and C) are those most used in creating subnets. Later, we'll see how a subnet uses these addresses. The IANA has outlined specific uses for multicast addresses within Internet Engineering Task Force (IETF) document RFC 5771. However, it hasn't designated a purpose or future plan for Class E addresses since it reserved the block in its 1989 document RFC 1112. Before IPv6, the Internet was filled with debate about whether the IANA should release Class E for general use.

Next, let's see how subnets work and find out who has those non-reserved IP addresses out on the Internet.

How DHCP Assigns Addresses

When you add a computer to a network, that computer uses a four-step process to get an IP address from DHCP:

☐ Discover – The computer sends out a broadcast message on the network, hoping to discover a DHCP service provider.

☐ Offer – Each DHCP provider hears the message, recognizes the unique hardware address of the computer, and sends a message back offering its services to that computer.

☐ Request – The computer selects a DHCP provider from its offerings and then sends a request to that provider asking for an IP address assignment.

☐ Acknowledge – The targeted DHCP provider acknowledges the request and issues an IP address to the computer that doesn't match any other IP addresses currently active on the network

(5,079 symbols)

http://computer.howstuffworks.com/internet/basics/question549.htm

TEXT 10 WHAT IS WI-FI?

First, let's get a couple of points out of the way: Wi-Fi, which rhymes with the outdated term "hi-fi," has nothing in common with its soundalike. Hi-fi stood for "high-fidelity" and was used to describe a phonograph/radio system with excellent sound.

Wi-Fi, by comparison, does not stand for "wireless fidelity" and has nothing to do with sound. In fact, it really doesn't stand for anything!

It simply represents wireless networking technology that allows you to go on the Internet without having to plug in any cables.

There's an organization called the Wi-Fi Alliance that actually owns the Wi-Fi trademark and controls or dictates the technology behind it.

Wi-Fi is everywhere these days, from people's homes to airports, hotels, libraries and just about every other place where people use their computers or wireless devices (laptops, smartphones and iPads/tablets).

Here are the main advantages of setting up a wireless network in your home:

- You can "connect" any and every computer in your home to your network without having to string cables/wires throughout the house.
- That means you can go on the Internet in any room from a laptop, desktop or smartphone.
- You can set up an access password that allows a visitor to log in to your network and will keep others from logging in without your permission...or knowledge.
- All it takes is a small, affordable piece of hardware called a "router" and some time to get things working.

Some well-known brands of routers are Belkin, Linksys and Netgear. You'll find plenty of information on routers online.

The good news is, you can set up a wireless network in your home pretty easily and quickly these days. It starts with your computer and grows from there. Here are some of the things you should know as you start your own network:

Start with your main computer: A wireless network needs to be set up. That's right: Even though your network will eventually be "wireless," to set it up you'll need to use your existing physical connection to the Internet.

A router comes with special software that has to be loaded on your computer. The software sets up the connection needed between your computer, modem and Internet Service Provider&emdash; and once everything is ready to go, you'll be able to invite and allow others devices to join your wireless network.

Safety and security. The wireless network broadcasts over a small area, but it has no boundaries. A next-door neighbor could easily be aware of your wireless network. That is, anyone close by with a wireless-enabled device might be able to see that a wireless network is nearby. However, without the password you create, they will not be able to access it or use it.

There are also security settings (which come with the router software) that will prevent hackers from intercepting your signal.

Wireless with wires. One more thing: Your "wireless" router has couple of wires, at least two. One is the electrical cord for power; the other is a cable (typically an Ethernet cable) with a connector that looks like a large telephone jack and that plugs into your modem. (If your router is a modem too, it will connect to your computer.)

That may sound funny, but the "wireless" feature has to do with the devices that will be able to connect wirelessly with the router.

Modem: You probably have a modem now - it's an important part of a wireless network. You need your modem to connect your computer to the Internet, and you'll still need it because your router works with your modem (or, you'll be replacing your modem with a device that is both a modem AND router).

A "wireless" router. Just as a mailman delivers mail to different addresses on your street, the router, once it's set up, will deliver an Internet connection (back and forth) for any computer or device with "wireless" capability.

Generation gap. Most electronic/computer stores carry a selection of routers. As with other technologies (cell phones, computers), routers have gone through several "generations" of development. As of 2013, most stores advertised "wireless-N" routers as the most common generation, with "wireless-ac" advertised as the "next" generation. (Before "N" came Wireless-G and Wireless-B.) As you might guess, the new generation tends to be better and, in this case, faster.

You'll also see routers that come with the designation "2.4 GHz" and/or "5 GHz" (gigahertz), which are the radio wave signals a router emits. What? Radio waves? Yes! That's how the signal is sent throughout your house and received by compatible wireless devices. If a router is both 2.4 and 5 GHz, it will be called a "dual band" router.

As with anything else related to technology, it helps to do some research and ask plenty of questions of salespeople and people you know. With the wireless network fundamentals you just read about, you should be able find the wireless system that will work best for you.

(3,876 symbols)

http://whatismyipaddress.com/wi-fi

TEXT 11 HOW DOMAIN NAME SERVERS WORK

If you've ever used the Internet, it's a good bet that you've used the **Domain Name System**, or **DNS**, even without realizing it. DNS is a protocol within the set of standards for how computers exchange data on the Internet and on many private networks, known as the TCP/IP protocol suite. Its basic job is to turn a user-friendly domain name like "howstuffworks.com" into an Internet Protocol (IP) address like 70.42.251.42 that computers use to identify each other on the network. It's like your computer's GPS for the Internet.

Computers and other network devices on the Internet use an IP address to route your request to the site you're trying to reach. This is similar to dialing a phone number to connect to the person you're trying to call. Thanks to DNS, though, you don't have to keep your own address book of IP addresses. Instead, you just connect through a domain name server, also called a DNS server or name server, which manages a massive database that maps domain names to IP addresses.

Whether you're accessing a Web site or sending e-mail, your computer uses a DNS server to look up the domain name you're trying to access. The proper term for this process is DNS name resolution, and you would say that the DNS server resolves the domain name to the IP address. For example, when you enter "http://www.howstuffworks.com" in your browser, part of the network connection includes resolving the domain name "howstuffworks.com" into an IP address, like 70.42.251.42, for HowStuffWorks' Web servers.

You can always bypass a DNS lookup by entering 70.42.251.42 directly in your browser (give it a try). However, you're probably more likely to remember "howstuffworks.com" when you want to return later. In addition, a Web site's IP address can change over time, and some sites associate multiple IP addresses with a single domain name.

Without DNS servers, the Internet would shut down very quickly. But how does your computer know what DNS server to use? Typically, when you connect to your home network, Internet service provider (ISP) or WiFi network, the modem or router that assigns your computer's network address also sends some important network configuration information to your computer or mobile device. That configuration includes one or more DNS servers that the device should use when translating DNS names to IP address.

So far, you've read about some important DNS basics. The rest of this article dives deeper into domain name servers and name resolution. It even

includes an introduction to managing your own DNS server. Let's start by looking at how IP addresses are structured and how that's important to the name resolution process.

Domain Names

If we had to remember the IP addresses of all our favorite Web sites, we'd probably go nuts! Human beings are just not that good at remembering strings of numbers. We are good at remembering words, however, and that is where domain names come in. You probably have hundreds of domain names stored in your head, such as:

- howstuffworks.com – our favorite domain name
- google.com – one of the most used domain names in the world
- mit.edu – a popular EDU name
- bbc.co.uk – a three-part domain name using the country code UK

You'll recognize domain names as having strings of characters separated by dots (periods). The last word in a domain name represents a top-level domain. These top-level domains are controlled by the IANA in what's called the Root Zone Database, which we'll examine more closely later. The following are some common top-level domains:

- COM – commercial Web sites, though open to everyone
- NET – network Web sites, though open to everyone
- ORG – non-profit organization Web sites, though open to everyone
- EDU – restricted to schools and educational organizations
- MIL – restricted to the U.S. military
- GOV – restricted to the U.S. government
- US, UK, RU and other two-letter country codes – each is assigned to a domain name authority in the respective country

In a domain name, each word and dot combination you add before a top-level domain indicates a level in the domain structure. Each level refers to a server or a group of servers that manage that domain level. For example, "howstuffworks" in our domain name is a second-level domain off the COM top-level domain. An organization may have a hierarchy of sub-domains further organizing its Internet presence, like "bbc.co.uk" which is the BBC's domain under CO, an additional level created by the domain name authority responsible for the UK country code.

The left-most word in the domain name, such as www or mail, is a host name. It specifies the name of a specific machine (with a specific IP address) in a domain, typically dedicated to a specific purpose. A given domain can potentially contain millions of host names as long as they're all unique to that domain.

Because all of the names in a given domain need to be unique, there has to be some way to control the list and makes sure no duplicates arise. That's where registrars come in. A registrar is an authority that can assign domain names directly under one or more top-level domains and register them with InterNIC, a service of ICANN, which enforces uniqueness of domain names across the Internet. Each domain registration becomes part of a central domain registration database known as the whois database. Network Solutions, Inc. (NSI) was one of the first registrars, and today companies like GoDaddy.com offer domain registration in addition to many other Web site and domain management services. [source: InterNIC]

(4,407 symbols)

http://computer.howstuffworks.com/dns.htm

TEXT 12 WHAT IS 3D PRINTING?

3D printing or additive manufacturing is a process of making three dimensional solid objects from a digital file.

The creation of a 3D printed object is achieved using additive processes. In an additive process an object is created by laying down successive layers of material until the object is created. Each of these layers can be seen as a thinly sliced horizontal cross-section of the eventual object.

How does 3D printing work?

It all starts with making a virtual design of the object you want to create. This virtual design is for instance a CAD (Computer Aided Design) file. This CAD file is created using a 3D modeling application or with a 3D scanner (to copy an existing object). A 3D scanner can make a 3D digital copy of an object.

3D scanners

3D scanners use different technologies to generate a 3D model. Examples are: time-of-flight, structured / modulated light, volumetric scanning and many more.

Recently, companies like Microsoft and Google enabled their hardware to perform 3D scanning, for example Microsoft's Kinect. In the near future digitizing real objects into 3D models will become as easy as taking a picture. Future versions of smartphones will probably have integrated 3D scanners. Currently, prices of 3D scanners range from expensive professional industrial devices to $30 DIY scanners anyone can make at home.

3D modeling software

3D modeling software also comes in many forms. There's industrial grade software that costs thousands a year per license, but also free open source software, like Blender, for instance. When you have a 3D model, the next step is to prepare it in order to make it 3D printable.

From 3D model to 3D Printer

You will have to prepare a 3D model before it is ready to be 3D printed. This is what they call slicing. Slicing is dividing a 3D model into hundreds or thousands of horizontal layers and needs to be done with software. Sometimes a 3D model can be sliced from within a 3D modeling software application. It is also possible that you are forced to use a certain slicing tool for a certain 3D printer. When the 3D model is sliced, you are ready to feed it to your 3D printer. This can be done via USB, SD or wifi. It really depends on what brand and type 3D Printer you have. When a file is uploaded in a 3D printer, the object is ready to be 3D printed layer by layer. The 3D printer reads every slice (2D image) and creates a three dimensional object.

Processes and technologies

Not all 3D printers use the same technology. There are several ways to print and all those available are additive, differing mainly in the way layers are build to create the final object. Some methods use melting or softening material to produce the layers. Selective laser sintering (SLS) and fused deposition modeling (FDM) are the most common technologies using this way of 3D printing. Another method is when we talk about curing a photo-reactive resin with a UV laser or another similar power source one layer at a time. The most common technology using this method is called stereolithography (SLA).

To be more precise: since 2010, the American Society for Testing and Materials (ASTM) group "ASTM F42 – Additive Manufacturing", developed a set of standards that classify the Additive Manufacturing processes into 7 categories according to Standard Terminology for Additive Manufacturing Technologies. These seven processes are:

1. Vat Photopolymerisation
2. Material Jetting
3. Binder Jetting
4. Material Extrusion
5. Powder Bed Fusion
6. Sheet Lamination
7. Directed Energy Deposition

Vat Photopolymerisation

A 3D printer based on the Vat Photopolymerisation method has a container filled with photopolymer resin which is then hardened with a UV light source.

The most commonly used technology in this processes is Stereolithography (SLA). This technology employs a vat of liquid ultraviolet curable photopolymer resin and an ultraviolet laser to build the object's layers one at a time. For each layer, the laser beam traces a cross-section of the part pattern on the surface of the liquid resin. Exposure to the ultraviolet laser light cures and solidifies the pattern traced on the resin and joins it to the layer below.

After the pattern has been traced, the SLA's elevator platform descends by a distance equal to the thickness of a single layer, typically 0.05 mm to 0.15 mm (0.002″ to 0.006″). Then, a resin-filled blade sweeps across the cross section of the part, re-coating it with fresh material. On this new liquid surface, the subsequent layer pattern is traced, joining the previous layer. The complete three dimensional object is formed by this project. Stereolithography requires the use of supporting structures which serve to attach the part to the elevator platform and to hold the object because it floats in the basin filled with liquid resin. These are removed manually after the object is finished.

This technique was invented in 1986 by Charles Hull, who also at the time founded the company, 3D Systems.

Other technologies using Vat Photopolymerisation are the new ultrafast Continuous Liquid Interface Production or CLIP and marginally used older Film Transfer Imaging and Solid Ground Curing.

Material Jetting

In this process, material is applied in droplets through a small diameter nozzle, similar to the way a common inkjet paper printer works, but it is applied layer-by-layer to a build platform making a 3D object and then hardened by UV light.

Binder Jetting

With binder jetting two materials are used: powder base material and a liquid binder. In the build chamber, powder is spread in equal layers and binder is applied through jet nozzles that "glue" the powder particles in the shape of a programmed 3D object. The finished object is "glued together" by binder remains in the container with the powder base material. After the print is finished, the remaining powder is cleaned off and used for 3D printing the next object. This technology was first developed at the

Massachusetts Institute of Technology in 1993 and in 1995 Z Corporation obtained an exclusive license.

Material Extrusion

The most commonly used technology in this process is Fused deposition modeling (FDM)

The FDM technology works using a plastic filament or metal wire which is unwound from a coil and supplying material to an extrusion nozzle which can turn the flow on and off. The nozzle is heated to melt the material and can be moved in both horizontal and vertical directions by a numerically controlled mechanism, directly controlled by a computer-aided manufacturing (CAM) software package. The object is produced by extruding melted material to form layers as the material hardens immediately after extrusion from the nozzle. This technology is most widely used with two plastic filament material types: ABS (Acrylonitrile Butadiene Styrene) and PLA (Polylactic acid) but many other materials are available ranging in properties from wood filed, conductive, flexible etc.

FDM was invented by Scott Crump in the late 80's. After patenting this technology he started the company Stratasys in 1988. The software that comes with this technology automatically generates support structures if required. The machine dispenses two materials, one for the model and one for a disposable support structure.

The term fused deposition modeling and its abbreviation to FDM are trademarked by Stratasys Inc. The exactly equivalent term, fused filament fabrication (FFF), was coined by the members of the RepRap project to give a phrase that would be legally unconstrained in its use.

Powder Bed Fusion

The most commonly used technology in this processes is Selective laser sintering (SLS).

This technology uses a high power laser to fuse small particles of plastic, metal, ceramic or glass powders into a mass that has the desired three dimensional shape. The laser selectively fuses the powdered material by scanning the cross-sections (or layers) generated by the 3D modeling program on the surface of a powder bed. After each cross-section is scanned, the powder bed is lowered by one layer thickness. Then a new layer of material is applied on top and the process is repeated until the object is completed.

All untouched powder remains as it is and becomes a support structure for the object. Therefore there is no need for any support structure which is an advantage over SLS and SLA. All unused powder can be used for the

next print. SLS was developed and patented by Dr. Carl Deckard at the University of Texas in the mid-1980s, under sponsorship of DARPA.

Sheet Lamination

Sheet lamination involves material in sheets which is bound together with external force. Sheets can be metal, paper or a form of polymer. Metal sheets are welded together by ultrasonic welding in layers and then CNC milled into a proper shape. Paper sheets can be used also, but they are glued by adhesive glue and cut in shape by precise blades. A leading company in this field is Mcor Technologies.

(7,190 symbols)

http://3dprinting.com/what-is-3d-printing/

TEXT 13 WHAT IS GIS?

This is probably the most asked question posed to those in the Geographic Information Systems (GIS) field and is probably the hardest to answer in a succinct and clear manner. GIS is a technological field that incorporates geographical features with tabular data in order to map, analyze, and assess real-world problems. The key word to this technology is Geography – this means that some portion of the data is spatial. In other words, data that is in some way referenced to locations on the earth. Coupled with this data is usually tabular data known as attribute data. Attribute data can be generally defined as additional information about each of the spatial features. An example of this would be schools. The actual location of the schools is the spatial data. Additional data such as the school name, level of education taught, student capacity would make up the attribute data. It is the partnership of these two data types that enables GIS to be such an effective problem solving tool through spatial analysis.

GIS operates on many levels. On the most basic level, geographic information systems technology is used as computer cartography, that is for straight forward mapping. The real power of GIS is through using spatial and statistical methods to analyze attribute and geographic information. The end result of the analysis can be derivative information, interpolated information or prioritized information.

GIS Versus Geospatial

There is an increasing trend to use the term geospatial instead of GIS. What is the difference between geospatial and GIS? Although some may use the terms interchangeably, there is a distinct difference between the

two in that GIS refers more narrowly to the traditional definition of using layers of geographic data to produce spatial analysis and derivative maps. Geospatial is more broadly use to refer to all technologies and applications of geographic data. For example, popular social media sites such as Foursquare and Facebook use "check-ins" that allow their users the ability to geographically tag their statuses. While those applications are considered to be geospatial, they don't fall underneath the stricter definition of what makes up a geographic information system.

GIS has already affected most of us in some way without us even realizing it. If you've ever used an Internet mapping program to find directions, congratulations, you've personally used GIS. The new supermarket chain on the corner was probably located using GIS to determine the most effective place to meet customer demand.

Uses of GIS

There are numerous ways in which this technology can be used. The most common ones are:

1. Management of resources
2. Investigations of the earth's surface that is scientific in nature
3. Archeological uses
4. Planning of locations and management of assets
5. Urban & regional planning
6. Criminology matters
7. An Impact assessment of the environment
8. The assessment and eventual development of infrastructure
9. Studies of the demographics of an area plus its population
10. Analysis with regards to engineering

Some of the common instances where you will find the GIS in use include:

1. Emergency response teams normally use GIS when they want to collect logistics with regards to how they will move in times of natural disasters.

2. The system also comes in handy when authorities want to discover any potential wetlands that need to be protected from the harmful effects brought about by pollution.

3. Companies also take advantage of the GIS so that they may be able to choose a strategic market location that has not yet been saturated by other competitors in the particular niche industry.

4. Management personnel use this system also so that they can be able to locate areas that are bound to suffer from catastrophes with regards to the infrastructure that is in place there.

5. Any potential spread of diseases & other such like pandemic are usually limited by the use of the GIS since the patterns of their occurrence is predicted in sufficient time.

The Development of GIS

One of the most famous early examples of spatial analysis can be traced back to London in the year 1854 when Dr. John Snow was able to predict the occurrence of cholera outbreak. Thanks to the study that Snow released, officials from the government were able to determine the cause of the disease; which was contaminated water from one of the major pumps. The map that Snow came up with was very interesting in that it had the capability of analyzing the phenomena relating to their geographical positions and this was the first time the world was witnessing this. Photozincography was developed in the earlier years of the 1900s and this enabled the maps to be divided into various layers as required. In the initial stages, the process of drawing these maps was lengthy since it involved free hand but this changed later on with the introduction of the computer.

The first GIS was created by Dr. Roger Tomlinson and then introduced in the early 1960s in Canada. During its inception, this system was mainly meant for collecting, storing and then analyzing the capability & potential which the land in the rural areas had. Prior to this, mapping by the use of computers was being used for such cases but this is a method that had numerous limitations associated to it. By the end of the 80s period, the use of GIS had already become popular in other related fields which is why it led to a spur in the growth of the industrial sector. Recently, designers came up with open source software for GIS so that the brilliant technology can be enhanced in a much simpler manner while being made available to all.

Components of GIS

The next step in understanding GIS is to look at each area and how they work together. These components are:

- Hardware
- Software
- Data
- People

Hardware

Hardware comprises the equipment needed to support the many activities needed for geospatial analysis ranging from data collection to data analysis. The central piece of equipment is the workstation, which runs the GIS software and is the attachment point for ancillary equipment. Data collection efforts can also require the use of a digitizer for conversion of hard copy data to digital data and a GPS data logger to collect data in the field. The use of handheld field technology is also becoming an important data collection tool in GIS. With the advent of web mapping, web servers have also become an important piece of equipment.

Software

Different types of software are important. Central to this is the GIS application package. Such software is essential for creating, editing and analyzing spatial and attribute data, therefore these packages contain a myriad of geospatial functions inherent to them. Extensions or add-ons are software that extends the capabilities of the GIS software package. Component GIS software is the opposite of application software. Component GIS seeks to build software applications that meet a specific purpose and thus are limited in their spatial analysis capabilities. Utilities are stand-alone programs that perform a specific function. For example, a file format utility that converts from on type of GIS file to another. There is also web GIS software that helps serve data and interactive maps through Internet browsers.

Data

Data is the core of any GIS. There are two primary types of data that are used in GIS: vector and raster data. A geodatabase is a database that is in some way referenced to locations on the earth. Geodatabases are grouped into two different types: vector and raster. Vector data is spatial data represented as points, lines and polygons. Raster data is cell-based data such as aerial imagery and digital elevation models. Coupled with this data is usually data known as attribute data. Attribute data generally defined as additional information about each spatial feature housed in tabular format. Documentation of GIS datasets is known as metadata. Metadata contains such information as the coordinate system, when the data was created, when it was last updated, who created it and how to contact them and definitions for any of the code attribute data.

People

Well-trained GIS professionals knowledgeable in spatial analysis and skilled in using GIS software are essential to the GIS process. There are

three factors to the people component: education, career path, and networking. The right education is key; taking the right combination of classes. Selecting the right type of GIS job is important. A person highly skilled in GIS analysis should not seek a job as a GIS developer if they haven't taken the necessary programming classes. Finally, continuous networking with other GIS professionals is essential for the exchange of ideas as well as a support community.

(7,159 symbols)

https://www.gislounge.com/what-is-gis/

TEXT 14 WHY PROGRAMMING?

You may already have used software, perhaps for word processing or spreadsheets, to solve problems. Perhaps now you are curious to learn how programmers write software. A *program* is a set of step-by-step instructions that directs the computer to do the tasks you want it to do and produce the results you want.

There are at least three good reasons for learning programming:

- Programming helps you understand computers. The computer is only a tool. If you learn how to write simple programs, you will gain more knowledge about how a computer works.
- Writing a few simple programs increases your confidence level. Many people find great personal satisfaction in creating a set of instructions that solve a problem.
- Learning programming lets you find out quickly whether you like programming and whether you have the analytical turn of mind programmers need. Even if you decide that programming is not for you, understanding the process certainly will increase your appreciation of what programmers and computers can do.

A set of rules that provides a way of telling a computer what operations to perform is called a programming language. There is not, however, just one programming language; there are many. In this chapter you will learn about controlling a computer through the process of programming. You may even discover that you might want to become a programmer.

An important point before we proceed: You will not be a programmer when you finish reading this chapter or even when you finish reading the final chapter. Programming proficiency takes practice and training beyond the scope of this book. However, you will become acquainted with how programmers develop solutions to a variety of problems.

What Programmers Do

In general, the programmer's job is to convert problem solutions into instructions for the computer. That is, the programmer prepares the instructions of a computer program and runs those instructions on the computer, tests the program to see if it is working properly, and makes corrections to the program. The programmer also writes a report on the program. These activities are all done for the purpose of helping a user fill a need, such as paying employees, billing customers, or admitting students to college.

The programming activities just described could be done, perhaps, as solo activities, but a programmer typically interacts with a variety of people. For example, if a program is part of a system of several programs, the programmer coordinates with other programmers to make sure that the programs fit together well. If you were a programmer, you might also have coordination meetings with users, managers, systems analysts, and with peers who evaluate your work-just as you evaluate theirs.

Let us turn to the programming process.

The Programming Process

Developing a program involves steps similar to any problem-solving task. There are five main ingredients in the programming process:

1. Defining the problem
2. Planning the solution
3. Coding the program
4. Testing the program
5. Documenting the program

Let us discuss each of these in turn.

1. Defining the Problem

Suppose that, as a programmer, you are contacted because your services are needed. You meet with users from the client organization to analyze the problem, or you meet with a systems analyst who outlines the project. Specifically, the task of defining the problem consists of identifying what it is you know (input-given data), and what it is you want to obtain (output-the result). Eventually, you produce a written agreement that, among other things, specifies the kind of input, processing, and output required. This is not a simple process.

2. Planning the Solution

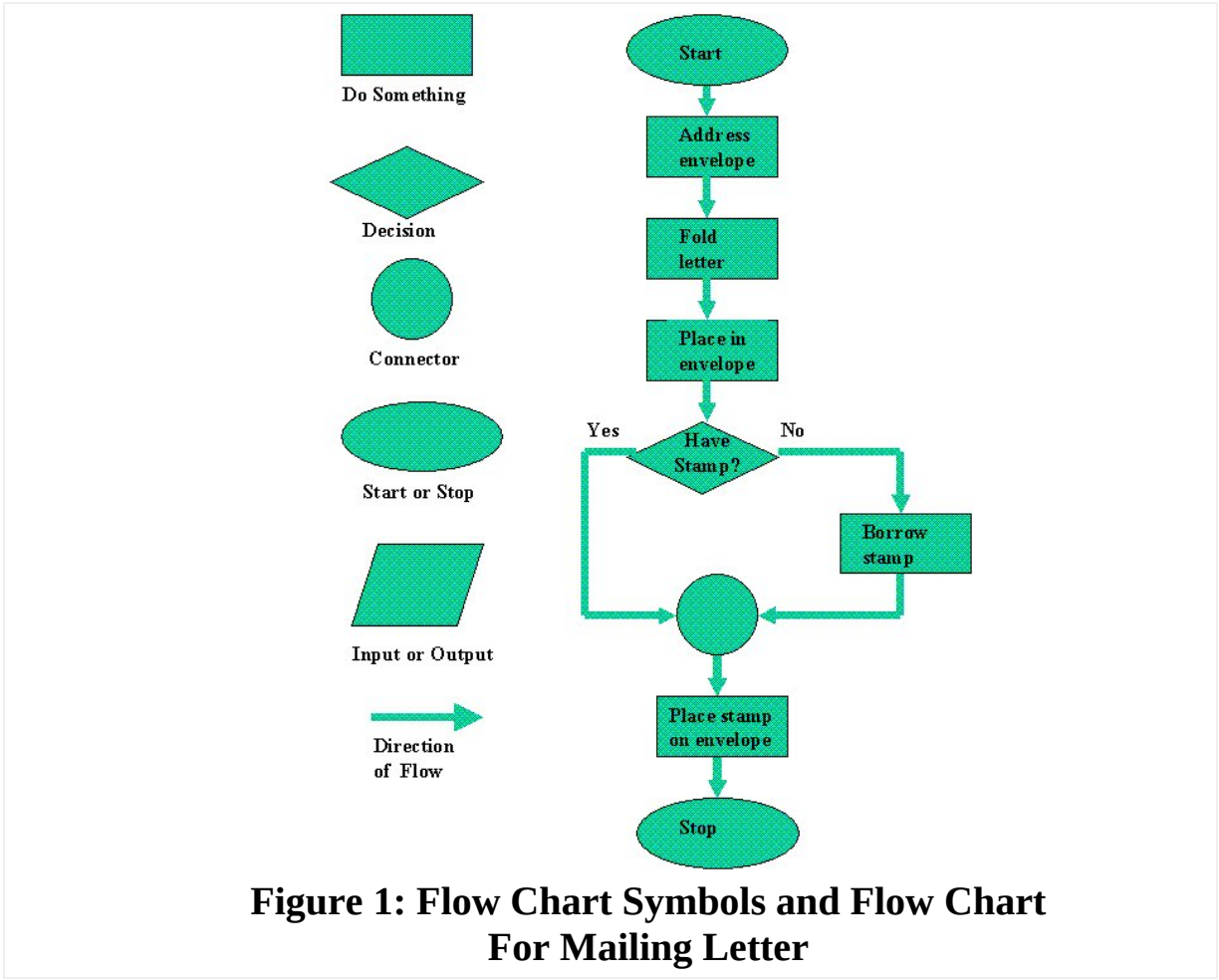


Figure 1: Flow Chart Symbols and Flow Chart For Mailing Letter

Two common ways of planning the solution to a problem are to draw a flowchart and to write pseudocode, or possibly both. Essentially, a flowchart is a pictorial representation of a step-by-step solution to a problem. It consists of arrows representing the direction the program takes and boxes and other symbols representing actions. It is a map of what your program is going to do and how it is going to do it. The American National Standards Institute (ANSI) has developed a standard set of flowchart symbols. Figure 1 shows the symbols and how they might be used in a simple flowchart of a common everyday act-preparing a letter for mailing.

Pseudocode is an English-like nonstandard language that lets you state your solution with more precision than you can in plain English but with less precision than is required when using a formal programming language. Pseudocode permits you to focus on the program logic without having to be concerned just yet about the precise syntax of a particular programming language. However, pseudocode is not executable on the computer. We will illustrate these later in this chapter, when we focus on language examples.

3. Coding the Program

As the programmer, your next step is to code the program-that is, to express your solution in a programming language. You will translate the logic from the flowchart or pseudocode-or some other tool-to a programming language. As we have already noted, a programming language is a set of rules that provides a way of instructing the computer what operations to perform. There are many programming languages: BASIC, COBOL, Pascal, FORTRAN, and C are some examples. You may find yourself working with one or more of these. We will discuss the different types of languages in detail later in this chapter.

Although programming languages operate grammatically, somewhat like the English language, they are much more precise. To get your program to work, you have to follow exactly the rules-the syntax-of the language you are using. Of course, using the language correctly is no guarantee that your program will work, any more than speaking grammatically correct English means you know what you are talking about. The point is that correct use of the language is the required first step. Then your coded program must be keyed, probably using a terminal or personal computer, in a form the computer can understand.

One more note here: Programmers usually use a text editor, which is somewhat like a word processing program, to create a file that contains the program. However, as a beginner, you will probably want to write your program code on paper first.

4. Testing the Program

Some experts insist that a well-designed program can be written correctly the first time. In fact, they assert that there are mathematical ways to prove that a program is correct. However, the imperfections of the world are still with us, so most programmers get used to the idea that their newly written programs probably have a few errors. This is a bit discouraging at first, since programmers tend to be precise, careful, detail-oriented people who take pride in their work. Still, there are many opportunities to introduce mistakes into programs, and you, just as those who have gone before you, will probably find several of them.

Eventually, after coding the program, you must prepare to test it on the computer. This step involves these phases:

- **Desk-checking.** This phase, similar to proofreading, is sometimes avoided by the programmer who is looking for a shortcut and is eager to run the program on the computer once it is written. However, with careful desk-checking you may discover several errors and possibly save yourself

time in the long run. In desk-checking you simply sit down and mentally trace, or check, the logic of the program to attempt to ensure that it is error-free and workable. Many organizations take this phase a step further with a walkthrough, a process in which a group of programmers-your peers-review your program and offer suggestions in a collegial way.

- **Translating.** A translator is a program that (1) checks the syntax of your program to make sure the programming language was used correctly, giving you all the syntax-error messages, called diagnostics, and (2) then translates your program into a form the computer can understand. A by-product of the process is that the translator tells you if you have improperly used the programming language in some way. These types of mistakes are called syntax errors. The translator produces descriptive error messages. For instance, if in FORTRAN you mistakenly write N=2 *(I+J))-which has two closing parentheses instead of one-you will get a message that says, "UNMATCHED PARENTHESES". (Different translators may provide different wording for error messages). Programs are most commonly translated by a *compiler*. A compiler translates your entire program at one time. The translation involves your original program, called a source module, which is transformed by a compiler into an object module. Prewritten programs from a system library may be added during the link/load phase, which results in a load module. The load module can then be executed by the computer.
- **Debugging.** A term used extensively in programming, debugging means detecting, locating, and correcting bugs (mistakes), usually by running the program. These bugs are logic errors, such as telling a computer to repeat an operation but not telling it how to stop repeating. In this phase you run the program using test data that you devise. You must plan the test data carefully to make sure you test every part of the program.

5. Documenting the Program

Documenting is an ongoing, necessary process, although, as many programmers are, you may be eager to pursue more exciting computer-centered activities. Documentation is a written detailed description of the programming cycle and specific facts about the program. Typical program documentation materials include the origin and nature of the problem, a brief narrative description of the program, logic tools such as flowcharts and pseudocode, data-record descriptions, program listings, and testing results. Comments in the program itself are also considered an essential part of documentation. Many programmers document as they code.

In a broader sense, program documentation can be part of the documentation for an entire system.

The wise programmer continues to document the program throughout its design, development, and testing. Documentation is needed to supplement human memory and to help organize program planning. Also, documentation is critical to communicate with others who have an interest in the program, especially other programmers who may be part of a programming team. And, since turnover is high in the computer industry, written documentation is needed so that those who come after you can make any necessary modifications in the program or track down any errors that you missed.

The Joys of the Field

Although many people make career changes into the computer field, few choose to leave it. In fact, surveys of computer professionals, especially programmers, consistently report a high level of job satisfaction. There are several reasons for this contentment. One is the challenge-most jobs in the computer industry are not routine. Another is security, since established computer professionals can usually find work. And that work pays well-you will probably not be rich, but you should be comfortable. The computer industry has historically been a rewarding place for women and minorities. And, finally, the industry holds endless fascination since it is always changing.

(9,291 symbols)

http://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading13.htm

TEXT 15 PROGRAMMING LANGUAGES

- **Levels of Language**

Programming languages are said to be "lower" or "higher," depending on how close they are to the language the computer itself uses (Os and 1s = low) or to the language people use (more English-like-high). We will consider five levels of language. They are numbered 1 through 5 to correspond to levels, or generations. In terms of ease of use and capabilities, each generation is an improvement over its predecessors. The five generations of languages are

1. Machine language
2. Assembly languages
3. High-level languages
4. Very high-level languages
5. Natural languages

Let us look at each of these categories.

Machine Language

Humans do not like to deal in numbers alone-they prefer letters and words. But, strictly speaking, numbers are what machine language is. This lowest level of language, machine language, represents data and program instructions as 1s and Os-binary digits corresponding to the on and off electrical states in the computer. Each type of computer has its own machine language. In the early days of computing, programmers had rudimentary systems for combining numbers to represent instructions such as add and compare. Primitive by today's standards, the programs were not convenient for people to read and use. The computer industry quickly moved to develop assembly languages.

Assembly languages

Today, assembly languages are considered very low level-that is, they are not as convenient for people to use as more recent languages. At the time they were developed, however, they were considered a great leap forward. To replace the Is and Os used in machine language, assembly languages use mnemonic codes, abbreviations that are easy to remember: A for Add, C for Compare, MP for Multiply, STO for storing information in memory, and so on. Although these codes are not English words, they are still- from the standpoint of human convenience-preferable to numbers (Os and 1s) alone. Furthermore, assembly languages permit the use of names- perhaps RATE or TOTAL-for memory locations instead of actual address numbers. just like machine language, each type of computer has its own assembly language.

The programmer who uses an assembly language requires a translator to convert the assembly language program into machine language. A translator is needed because machine language is the only language the computer can actually execute. The translator is an assembler program, also referred to as an assembler. It takes the programs written in assembly language and turns them into machine language. Programmers need not worry about the translating aspect; they need only write programs in assembly language. The translation is taken care of by the assembler.

Although assembly languages represent a step forward, they still have many disadvantages. A key disadvantage is that assembly language is detailed in the extreme, making assembly programming repetitive, tedious, and error prone. Assembly language may be easier to read than machine language, but it is still tedious.

High-Level Languages

The first widespread use of high-level languages in the early 1960s transformed programming into something quite different from what it had been. Programs were written in an English-like manner, thus making them more convenient to use. As a result, a programmer could accomplish more with less effort, and programs could now direct much more complex tasks.

These so-called third-generation languages spurred the great increase in data processing that characterized the 1960s and 1970s. During that time the number of mainframes in use increased from hundreds to tens of thousands. The impact of third-generation languages on our society has been enormous.

Of course, a translator is needed to translate the symbolic statements of a high-level language into computer-executable machine language; this translator is usually a compiler. There are many compilers for each language and one for each type of computer. Since the machine language generated by one computer's COBOL compiler, for instance, is not the machine language of some other computer, it is necessary to have a COBOL compiler for each type of computer on which COBOL programs are to be run. Keep in mind, however, that even though a given program would be compiled to different machine language versions on different machines, the source program itself-the COBOL version-can be essentially identical on each machine.

Some languages are created to serve a specific purpose, such as controlling industrial robots or creating graphics. Many languages, however, are extraordinarily flexible and are considered to be general-purpose. In the past the majority of programming applications were written in BASIC, FORTRAN, or COBOL-all general-purpose languages. In addition to these three, another popular high-level language is C, which we will discuss later.

Very High-Level Languages

Languages called very high-level languages are often known by their generation number, that is, they are called fourth-generation languages or, more simply, 4GLs.

Definition

Will the real fourth-generation languages please stand up? There is no consensus about what constitutes a fourth-generation language. The 4GLs are essentially shorthand programming languages. An operation that requires hundreds of lines in a third-generation language such as COBOL

typically requires only five to ten lines in a 4GL. However, beyond the basic criterion of conciseness, 4GLs are difficult to describe.

Characteristics

Fourth-generation languages share some characteristics. The first is that they make a true break with the prior generation-they are basically non-procedural. A procedural language tells the computer how a task is done: Add this, compare that, do this if something is true, and so forth-a very specific step-by-step process. The first three generations of languages are all procedural. In a nonprocedural language, the concept changes. Here, users define only what they want the computer to do; the user does not provide the details of just how it is to be done. Obviously, it is a lot easier and faster just to say what you want rather than how to get it. This leads us to the issue of productivity, a key characteristic of fourth-generation languages.

Productivity

Folklore has it that fourth-generation languages can improve productivity by a factor of 5 to 50. The folklore is true. Most experts say the average improvement factor is about 10-that is, you can be ten times more productive in a fourth-generation language than in a third-generation language. Consider this request: Produce a report showing the total units sold for each product, by customer, in each month and year, and with a subtotal for each customer. In addition, each new customer must start on a new page. A 4GL request looks something like this:

```
TABLE FILE SALES
SUM UNITS BY MONTH BY CUSTOMER BY PRODUCT
ON CUSTOMER SUBTOTAL PAGE BREAK
END
```

Even though some training is required to do even this much, you can see that it is pretty simple. The third-generation language COBOL, however, typically requires over 500 statements to fulfill the same request. If we define productivity as producing equivalent results in less time, then fourth-generation languages clearly increase productivity.

Downside

Fourth-generation languages are not all peaches and cream and productivity. The 4GLs are still evolving, and that which is still evolving cannot be fully defined or standardized. What is more, since many 4GLs are easy to use, they attract a large number of new users, who may then overcrowd the computer system. One of the main criticisms is that the new languages lack the necessary control and flexibility when it comes to planning how you want the output to look. A common perception of 4GLs

is that they do not make efficient use of machine resources; however, the benefits of getting a program finished more quickly can far outweigh the extra costs of running it.

Benefits

Fourth-generation languages are beneficial because

- They are results-oriented; they emphasize what instead of how.
- They improve productivity because programs are easy to write and change.
- They can be used with a minimum of training by both programmers and nonprogrammers.
- They shield users from needing an awareness of hardware and program structure.

It was not long ago that few people believed that 4GLs would ever be able to replace third-generation languages. These 4GL languages are being used, but in a very limited way.

Query Languages

A variation on fourth-generation languages are query languages, which can be used to retrieve information from databases. Data is usually added to databases according to a plan, and planned reports may also be produced. But what about a user who needs an unscheduled report or a report that differs somehow from the standard reports? A user can learn a query language fairly easily and then be able to input a request and receive the resulting report right on his or her own terminal or personal computer. A standardized query language, which can be used with several different commercial database programs, is Structured Query Language, popularly known as SQL. Other popular query languages are Query-by-Example, known as QBE, and Intellect.

Natural Languages

The word *"natural"* has become almost as popular in computing circles as it has in the supermarket. Fifth-generation languages are, as you may guess, even more ill-defined than fourth-generation languages. They are most often called natural languages because of their resemblance to the "natural" spoken English language. And, to the manager new to computers for whom these languages are now aimed, natural means human-like. Instead of being forced to key correct commands and data names in correct order, a manager tells the computer what to do by keying in his or her own words.

REPORT THE BASE SALARY, COMMISSIONS AND YEARS OF
SERVICE BROKEN DOWN BY STATE AND CITY FOR SALESCLERKS
IN NEW JERSEY AND MASSACHUSETTS.

You can hardly get closer to conversational English than that. Natural languages excel at easy data access. Indeed, the most common application for natural languages is interacting with databases.

Choosing a Language

How do you choose the language with which to write your program? There are several possibilities:

- In a work environment, your manager may decree that everyone on your project will use a certain language.
- You may use a certain language, particularly in a business environment, based on the need to interface with other programs; if two programs are to work together, it is easiest if they are written in the same language.
- You may choose a language based on its suitability for the task. For example, a business program that handles large files may be best written in the business language COBOL.
- If a program is to be run on different computers, it must be written in a language that is portable-suitable on each type of computer-so that the program need be written only once.
- You may be limited by the availability of the language. Not all languages are available in all installations or on all computers.
- The language may be limited to the expertise of the programmer; that is, the program may have to be written in a language the available programmer knows.
- Perhaps the simplest reason, one that applies to many amateur programmers, is that they know the language called BASIC because it came with-or was inexpensively purchased with-their personal computers.

(9,335 symbols)

http://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading13.htm

TEXT 16 MAJOR PROGRAMMING LANGUAGES

FORTRAN: The First High-Level Language

Developed by IBM and introduced in 1954, FORTRAN-for FORmula TRANslator-was the first high-level language. FORTRAN is a scientifically oriented language-in the early days use of the computer was primarily associated with engineering, mathematical, and scientific research tasks.

FORTRAN is noted for its brevity, and this characteristic is part of the reason why it remains popular. This language is very good at serving its

primary purpose, which is execution of complex formulas such as those used in economic analysis and engineering. Although in the past it was considered limited in regard to file processing or data processing, its capabilities have been greatly improved.

Not all programs are organized in the same way. Organization varies according to the language used. In many languages (such as COBOL), programs are divided into a series of parts. FORTRAN programs are not composed of different parts (although it is possible to link FORTRAN programs together); a FORTRAN program consists of statements one after the other. Different types of data are identified as the data is used. Descriptions for data records appear in format statements that accompany the READ and WRITE statements. Figure 5 shows a FORTRAN program and a sample output from the program.

COBOL: The Language of Business

```
IF SALES-AMOUNT IS GREATER THAN SALES-QUOTA
COMPUTE COMMISSION = MAX-RATE * SALES-AMOUNT
ELSE
COMPUTE COMMISSION = MIN-RATE * SALES-AMOUNT.
```

Once you understand programming principles, it is not too difficult to add COBOL to your repertoire. COBOL can be used for just about any task related to business programming; indeed, it is especially suited to processing alphanumeric data such as street addresses, purchased items, and dollar amounts-the data of business. However, the feature that makes COBOL so useful-its English-like appearance and easy readability-is also a weakness because a COBOL program can be incredibly verbose. A programmer seldom knocks out a quick COBOL program. In fact, there is hardly such a thing as a quick COBOL program; there are just too many program lines to write, even to accomplish a simple task. For speed and simplicity, BASIC, FORTRAN, and Pascal are probably better bets.

A COBOL program is divided into four parts called divisions. The identification division identifies the program by name and often contains helpful comments as well. The environment division describes the computer on which the program will be compiled and executed. It also relates each file of the program to the specific physical device, such as the tape drive or printer, that will read or write the file. The data division contains details about the data processed by the program, such as type of characters (whether numeric or alphanumeric), number of characters, and placement of decimal points. The procedure division contains the statements that give the computer specific instructions to carry out the logic of the program.

It has been fashionable for some time to criticize COBOL: It is old-fashioned, cumbersome, and inelegant. In fact, some companies, devoted to fast, nimble program development, are converting to the more trendy language C. But COBOL, with more than 30 years of staying power, is still famous for its clear code, which is easy to read and debug.

Pascal: The Language of Simplicity

Named for Blaise Pascal, the seventeenth-century French mathematician, Pascal was developed as a teaching language by a Swiss computer scientist, Niklaus Wirth, and first became available in 1971. Since that time it has become quite popular, first in Europe and now in the United States, particularly in universities and colleges offering computer science programs.

The foremost feature of Pascal is that it is simpler than other languages -it has fewer features and is less wordy than most. In addition to the popularity of Pascal in college computer science departments, the language has also made large inroads in the personal computer market as a simple yet sophisticated alternative to BASIC. Over the years new versions have improved on the original capabilities of Pascal. Today, Borland's Turbo Pascal leads the Pascal world because its designers eliminated most of the drawbacks of the original Pascal. Turbo Pascal is used by the business community and is often the choice of nonprofessional programmers who need to write their own programs.

Ada: Named for the Countess

Is any software worth over $25 billion? Not any more, according to Defense Department experts. In 1974 the U.S. Department of Defense had spent that amount on all kinds of software for a hodgepodge of languages for its needs. The answer to this problem turned out to be a new language called Ada-named for Countess Ada Lovelace, "the first programmer" (see Appendix B). Sponsored by the Pentagon, Ada was originally intended to be a standard language for weapons systems, but it has also been used successfully for commercial applications. Introduced in 1980, Ada has the support not only of the defense establishment but also of such industry heavyweights as IBM and Intel, and Ada is even available for some personal computers. Although some experts have said Ada is too complex, others say that it is easy to learn and that it will increase productivity. Indeed, some experts believe that it is by far a superior commercial language to such standbys as COBOL and FORTRAN.

Widespread use of Ada is considered unlikely by many experts. Although there are many reasons for this (the military services, for

instance, have different levels of enthusiasm for it), probably its size-which may hinder its use on personal computers-and complexity are the greatest barriers. Although the Department of Defense is a market in itself, Ada has not caught on to the extent that Pascal and C have, especially in the business community.

C, C++, Java, and Javascript

A language invented by Dennis Ritchie at Bell Labs in 1972, C produces code that approaches assembly language in efficiency while still offering high-level language features. C was originally designed to write systems software but is now considered a general-purpose language. C contains some of the best features from other languages, including Pascal. C compilers are simple and compact. A key attraction is that it is independent of the architecture of any particular machine, a fact that contributes to the portability of C programs. That is, a C program can be run on more than one type of computer after it has been compiled for that machine.

Although C is simple and elegant, it is not simple to learn. It was developed for gifted programmers, and the learning curve may be steep. Straightforward tasks may be solved easily in C, but complex problems require mastery of the language.

An interesting side note is that the availability of C on personal computers has greatly enhanced the value of personal computers for budding software entrepreneurs. A cottage software industry can use the same basic tool-the language C-used by established software companies such as Microsoft and Borland. Today C is has been replaced by its enhanced cousin, C++. C++ in turn is being challenged by web-aware languages like Java and Javascript, that look and act a lot like C++, but add features to support working with networked computers, among other things.

(5,983 symbols)

http://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading13.htm

TEXT 17 THE HISTORY OF PASCAL

Origins

Pascal grew out of ALGOL, a programming language intended for scientific computing. Meeting in Zurich, an international committee designed ALGOL as a platform-independent language. This gave them more more free rein in the features they could put into it, but also made it more difficult to write compilers for it. Those were the days when many

computers lacked hardware features that we now take for granted. The lack of compilers on many platforms, combined with its lack of pointers and many basic data types such as characters, led to ALGOL not being widely accepted. Scientists and engineers flocked to FORTRAN, a programming language which *was* available on many platforms. ALGOL mostly faded away except as a language for describing algorithms.

Wirth Invents Pascal

In the 1960s, several computer scientists worked on extending ALGOL. One of these was Dr. Niklaus Wirth of the Swiss Federal Institute of Technology (ETH-Zurich), a member of the original group that created ALGOL. In 1971, he published his specification for a highly-structured language which resembled ALGOL in many ways. He named it *Pascal* after the 17th-century French philosopher and mathematician who built a working mechanical digital computer.

Pascal is very data-oriented, giving the programmer the ability to define custom data types. With this freedom comes strict type-checking, which ensured that data types didn't get mixed up. Pascal was intended as a teaching language, and was widely adopted as such. Pascal is free-flowing, unlike FORTRAN, and reads very much like a natural language, making it very easy to understand code written in it.

UCSD Pascal

One of the things that killed ALGOL was the difficulty of creating a compiler for it. Dr. Wirth avoided this by having his Pascal compiler compile to an intermediate, platform-independent object code stage. Another program turned this intermediate code into executable code.

Prof. Ken Bowles at the University of California at San Diego (UCSD) seized on the opportunity this offered to adapt the Pascal compiler to the Apple II, the most popular microcomputer of the day. UCSD P-System became a standard, and was widely used at universities. This was aided by the low cost of Apple II's compared to mainframes, which were necessary at the time to run other languages such as FORTRAN. Its impact on computing can be seen in IBM's advertisements for its revolutionary Personal Computer, which boasted that the PC supported three operating systems: Digital Research's CP/M-86, Softech's UCSD P-system, and MicroSoft's PC-DOS.

Pascal Becomes Standard

By the early 1980's, Pascal has already become widely accepted at universities. Two things happened to make it even more popular.

First, the Educational Testing Service, the company which writes and administers the principal college entrance exam in the United States, decided to add a Computer Science exam to its Advanced Placement exams for high school students. For this exam, it chose the Pascal language. Because of this, secondary-school students as well as college students began to learn Pascal. Pascal remained the official language of the AP exams until 1999, when it was replaced by C++, which was quickly replaced by Java.

Second, a small company named Borland International came out with the Turbo Pascal compiler for the IBM Personal Computer. This compiler was truly revolutionary. It did take some shortcuts and made some modifications to standard Pascal, but these were minor and led to its greatest advantage: speed. Turbo Pascal compiled at a dizzying rate: several thousand lines a minute. At the time, the available compilers for the PC platform were slow and bloated. When Turbo Pascal came out, it was a breath of fresh air. Soon, Turbo Pascal became the *de facto* standard for programming on the PC. When computing magazines published source code for utility programs, it was usually in either assembly or Turbo Pascal.

At the same time, Apple came out with its Macintosh series of computers. As Pascal was the preeminent structured programming language of the day, Apple chose Pascal as the standard programming language for the Mac. When programmers received the API and example code for Mac programming, it was all in Pascal.

Extensions

From version 1.0 to 7.0 of Turbo Pascal, Borland continued to expand the language. One of the criticisms of the original version of Pascal was its lack of separate compilation for modules. Dr. Wirth even created a new programming language, Modula-2, to address that problem. Borland added this to Pascal with its units feature.

By version 7.0, many advanced features had been added. One of these was DPMI (DOS Protected Mode Interface), a way to run DOS programs in protected mode, gaining extra speed and breaking free of the 640K barrier for accessing memory under DOS. Turbo Vision, a text-based windowing system, allowed programmers to create great-looking interfaces in practically no time at all. Pascal even became object-oriented, as version 5.5 adopted the Apple Object Pascal extensions. When Windows 3.0 came out, Borland created Turbo Pascal for Windows, bringing the speed and

ease of Pascal to the graphical user interface. It seemed that Pascal's future was secure.

The World Changes

However, this was not so. In the 1970s, Dennis Ritchie and Brian Kernighan of AT&T Bell Laboratories created the C Programming Language. Ritchie then collaborated with Ken Thompson to design the UNIX operating system. AT&T had, at that time, a monopoly on telephone service in the United States, and was operating under a consent decree which included being banned from the computer business. AT&T thus gave away the operating system, with source code, to universities for free.

Thus, a whole generation of computer science students learned Pascal in the introductory programming courses, then learned C when they delved into operating systems. Slowly but surely, C began to filter into the computer programming world.

The killer, ironically enough, was object orientation and the move to Windows on the PC platform. Bjarne Stroustrop introduced object-orientation to most of the world when he created C++. Apple created Object Pascal to handle the buttons and windows and other naturally object-oriented elements of a windowing operating system. But for most programmers, the first thing that pops to mind when OOP is mentioned is *C++*.

At the same time, Microsoft Windows adopted C as its standard programming language. As object orientation and Windows took hold, the natural code migration path for Windows applications was C++.

Many colleges and universities moved away from Pascal, choosing C++, or the new Java, for their programming courses. Finally, the AP exam moved to C++, ending Pascal's dominance in high schools.

So Why Learn Pascal?

Despite its fading away as a *de facto* standard, Pascal is still extremely useful. C and C++ are very symbolic languages. Where Pascal chooses words (e.g. begin-end), C/C++ chooses symbols (**{-}**). Also, C and C++ are not strongly-typed languages. In Pascal, mixing types often led to an error. In C/C++, type-casting and pointer arithmetic is common, making it easy to crash programs and write in buffer overruns. When the AP exam switched to C++, only a subset of C++ was adopted. Many features, like arrays, were considered too dangerous for students, and ETS provided its own "safe" version of these features. Java corrects many of these problems of C++, at the cost of slow execution.

Another reason: speed and size. The Borland Pascal compiler is still lightning- fast. Borland has revitalized Pascal for Windows with Delphi, a Rapid-Application-Development environment. Instead of spending several hours writing a user interface for a Windows program in C/C++, you could do it in ten minutes with Delphi's graphical design tools. Delphi is to Pascal what Visual BASIC did to BASIC.

Also, Pascal remains preferred at many universities, especially overseas where many students are exposed to computers at school rather than at home. In addition, Pascal was well-suited for teaching programming, and remains so. There's less overhead and fewer ways for a student to get a program into trouble. For teaching simple procedural programming, Pascal remains the top choice.

Thus, even after C, C++, and Java took over the programming world, Pascal retains a niche in the market. Many small-scale freeware, shareware, and open-source programs are written in Pascal/Delphi. So enjoy learning it while it lasts. It's a great introduction to computer programming. It's not scary like C, dangerous like C++, or abstract like Java. In another twenty years, you'll be one of the few computer programmers to know and appreciate Pascal.

(7,023 symbols)

http://www.bio.vu.nl/thb/course/comp/pascal/history.html

TEXT 18 OBJECT-ORIENTED PROGRAMMING

What is object-orientation in web development, and why is it important?

At its core, it’s a logic – one we use in daily life. We naturally think of things as objects with attributes and behaviors, and that determines how we interact with them. It’s interacting in the abstract, and it’s why OOP can boost speed and efficiency.

What is abstract interaction?

If you want to change the television channel from your seat, you use a remote control. That remote control is an object with a number of attributes and behaviors hidden inside of it. Without an understanding of those hidden attributes – the microchips, wiring, etc. – you still know and expect that pressing a button will perform that particular function. You’ve interacted with the remote control in *the abstract*, skipping the steps the remote was designed to carry out. That’s the beauty of OOP – the focus is

on how the objects behave, not the code required to tell them how to behave.

So, what are objects?

A car is an example of a complex object, with many attributes. We don't need to understand all of its internal mechanics, what kind of engine it has, how the gas makes it run, or even where the gas came from in order to know how to interact with it. The car's behaviors have been made simple for us through object-oriented logic: put the key in the ignition, and the car turns on and gets us where we need to go. The attributes that make this possible – all of the car's parts, electronics, and engineering – are a "package" we don't need to break down in order to understand.

Apply this to software building, and it allows developers to break down big, complicated projects into compartmentalized objects, program them to have attributes and behaviors, then essentially set them aside and focus on programming how the objects interact – a higher level of thinking that makes writing code less linear and more efficient. Modern, high-level languages like Python and Ruby are perfect examples of OOP. The fact that they're able to be so streamlined gets right to the heart of OOP logic.

Object-oriented programming & back-end development

What is object-oriented programming in terms of how a site is built? OOP defines most modern server-side scripting languages, which are the languages back-end developers use to write software and database technology. This behind-the-scenes, server-side technology tells a website or web application how to behave, and also builds the architecture for a site to interact with its database. That scaffolding is how data is delivered and processed, effectively making it *the brain* of a website. And that's where object-oriented logic comes into play.

If a website's brain uses object-oriented logic, it's designed to think of data as objects. It affects how a site is built from the ground up, how data is organized, how later growth and maintenance of the site will occur, and more.

Benefits of object-oriented technology include:

- Ease of software design
- Productivity
- Easy testing, debugging, and maintenance
- It's reusable
- More thorough data analysis, less development time, and more accurate coding, thanks to OOP's inheritance method

- Data is safe and secure, with less data corruption, thanks to hiding and abstraction
- It's sharable (classes are reusable and can be distributed to other networks)

The building blocks of object-oriented programming

Objects are central to OOP, but they're not the only moving part. Here's a closer look at the other building blocks, and how they work in tandem to create back-end code that houses, moves, and manipulates data from a database into a usable web application.

1. **Objects:** An object is the core unit of OOP. Objects are uniquely named and represent an instance of a class. Each object houses different states (attributes), and shared behaviors, called methods. For example, a Prius is an object in the class of "cars," in a subclass of "hybrid cars". Its attributes include anything from the number of doors it has to how its electric component is charged. It's similar to other cars by its behavior – it drives – but its attributes are what set it apart.
2. **Classes:** A class is a blueprint for how an object is built, as well as being a sort of "parent category" for objects. Using the previous example, a class dictates the concept of a car – four wheels, an engine, a body, brakes, etc. It allows certain set criteria to be passed down to all objects in the class. All varieties of cars behave relatively the same on a basic level, but its their attributes and methods that make them unique.
3. **Inheritance:** This is an important aspect of OOP, hinted at above. By deriving classes from parent classes, behaviors can be passed down to objects, then more complicated attributes can be added the deeper you go. For example, breaking a car into subclasses (car → sports car → V8 sports car) makes it possible to layer in more features without starting from scratch.
4. **Abstraction & Encapsulation**: This describes how attributes are housed and hidden within an object – including its data. Objects are designed to only reveal the necessary data, allowing software to interact with the object on a higher level. It's equal parts security and simplicity. In the case of car parts, by safely stowing them within the body of an assembled car, things are less likely to get broken, and users can interact with the big picture: pressing the gas means go, no questions asked.

Procedural languages vs. Object-oriented languages

Procedural programming (via languages like ColdFusion) is code that is broken into "procedures" – it's a different way of thinking about how code interacts with data that's more linear. Procedures are functional bits

of code that interact with and change data, like little machines that gather input, process it, then deliver output. With OOP, however, data and functions (attributes and methods) are bundled together within the object. This prevents the need for any shared or global data with OOP, which is a core difference between the two approaches.

Traditional procedural languages like C and Pascal require you to think in terms of the computer rather than thinking in terms of the problem you're trying to solve. For less complicated applications, procedural languages offer ease and transparency that bundled objects don't always allow – something that can make it more difficult for programmers to analyze smaller bits of code on the tail-end of the development process.

When it comes to creating reusable components in software, OOP is the clear winner. Reusability leads to efficiency, simplifying programming and creating "shortcuts" to software design.

(5,312 symbols)

https://www.upwork.com/hiring/development/object-oriented-programming/

TEXT 19 THE HISTORY OF DATABASE PROCESSING

Database processing was originally used in major corporations and large organizations as the basis of large transaction-processing systems. Later, as microcomputers gained popularity, database technology migrated to micros and was used for single-user, personal database applications. Next, as micros were connected together in work groups, database technology moved to the workgroup setting. Finally, databases are being used today for Internet and intranet applications.

1960: The Organizational Context

The initial application of database technology was to resolve problems with the file-processing systems. In the mid-1960s, large corporations were producing data at phenomenal rates in file-processing systems, but the data were becoming difficult to manage, and new systems were becoming increasingly difficult to develop. Furthermore, management wanted to be able to relate the data in one file system to those in another.

The limitations of file processing prevented the easy integration of data. Database technology, however, held out the promise of a solution to these problems, and so large companies began to develop organizational databases, Companies centralized their operational data, such as orders,

inventory, and accounting data, in these databases. The applications were primarily organization-wide, transaction processing systems.

At first, when the technology was new, database applications were difficult to develop, and there were many failures. Even those applications that were successful were slow and unreliable: The computer hardware could not handle the volume of transactions quickly; the developers had not yet discovered more efficient ways to store and retrieve data; and the programmers were still new at accessing databases, and sometimes their programs did not work correctly.

Companies found another disadvantage of database processing: vulnerability. If a file-processing system fails, only that particular application will be out of commission. But if the database fails, all of its dependent applications will be out of commission.

Gradually, the situation improved. Hardware and software engineers learned how to build systems powerful enough to support many concurrent users and fast enough to keep up with the daily workload of transactions. New ways of controlling, protecting, and backing up the database were devised. Standard procedures for database processing evolved, and programmers learned how to write more efficient and more maintainable code. By the mid-1970s, databases could efficiently and reliably process organizational applications. Many of those applications are still running today, more than 25 years after their creation!

1970: The Relational Model

In 1970, E.F. Codd published a landmark paper in which he applied concepts from a branch of mathematics called relational algebra to the problem of storing large amounts of data. Codd's paper started a movement in the database community that in a few years led to the definition of the relational database model. This model is a particular way of structuring and processing a database.

Benefits of the relational model.

The advantage of the relational model is that data are stored in a way that minimizes duplicated data and eliminates certain types of processing errors that can occur when data are stored in other ways. Data are stored as tables, with rows and columns.

According to the relational model, not all tables are equally desirable. Using a process called *normalization* a table that is not desirable can be changed into two or more that are.

Another key advantage of the relational model is that columns contain data that relate one row to another. This makes the relationships among rows visible to the user.

At first, it was thought that the relational model would enable users to obtain information from databases without the assistance of MIS professionals. Part of the rationale of this idea was that tables are simple constructs that are intuitively understandable. Additionally, since the relationships are stored in the data, the users would be able to combine rows when necessary.

It turned out that this process was too difficult for most users. Hence, the promise of the relational model as a means for non-specialists to access a database was never realized. In retrospect, the key benefit of the relational model has turned out to be that it provides a standard way for specialists (like you!) to structure and process a database.

Resistance to the relational model.

Initially the relational model encountered a good deal of resistance. Relational database systems require more computer resources, and so at first they were much slower than the systems based on earlier database models. Although they were easier to use, the slow response time was often unacceptable. To some extent, relational DBMS products were impractical until the 1980s, when faster computer hardware was developed and the price-performance ratio of computers fell dramatically.

The relational model also seemed foreign to many programmers, who were accustomed to writing programs in which they processed data one record at a time. But relational DBMS products process data most naturally an entire table at a time. Accordingly, programmers had to learn a new way to think about data processing.

Because of these problems, even though the relational model had many advantages, it did not gain true popularity until computers became more powerful. In particular, as microcomputers entered the scene, more and more CPU cycles could be devoted to a single user. Such power was a boon to relational DBMS products and set the stage for the nest major database development.

1980: Object-Oriented DBMS and Microcomputer DBMS Products

In 1979, a small company called Ashton-Tate introduced a microcomputer product, dBase II (pronounced “d base two”), and called it a relational DBMS. In and exceedingly successful promotional tactic, Ashton-Tate distributed – nearly free of charge – more than 100,000 copies

of its product to purchasers of the then new Osborne microcomputers. Many of the people who bought these computers were pioneers in the microcomputer industry. They began to invent microcomputer applications using dBase, and the number of dBase applications grew quickly. As a result, Ashton-Tate became one of the first major corporations in the microcomputer industry. Later, Ashton-Tate was purchased by Borland, which now sells the dBase line of products.

The success of this product, however, confused and confounded the subject of database processing. The problem was this: According to the definition prevalent in the late 1970s, dBase II was neither a DBMS nor relational. In fact, it was a programming language with generalized file-processing (not database-processing) capabilities. The systems that were developed with dBase II appeared much more like those in Figure 1-10 that the ones in Figure 1-9. the million or so users of dBase II thought they were using a relational DBMS when, in fact, they were not.

Thus, the terms ‘database management system and relational database were used loosely at the start of the microcomputer boom. Most of the people who were processing a microcomputer database were really managing files and were not receiving the benefits of database processing, although they did not realize it. Today, the situation has changed as the microcomputer marketplace has become more mature and sophisticated. dBase 5 and the dBase products that followed it are truly relational DBMS products.

Although dBase did pioneer the application of database technology on microcomputers, at the same time other vendors began to move their products from the mainframe to the microcomputer. Oracle, Focus, and Ingress are three examples of DBMS products that were ported down to microcomputers. They are truly DBMS programs, and most would agree that they are truly relational as well. In addition, other vendors developed new relational DBMS products especially for micros. Paradox, Revelation, MDBS, Helix, and a number of other products fall into this category.

One impact of the move of database technology to the micro was the dramatic improvement in DBMS user interfaces. Users of microcomputer systems are generally not MIS professionals, and they will not put up with the clumsy and awkward user interfaces common on mainframe DBMS products. Thus, as DBMS products were devised for micros, user interfaces had to be simplified and made easier to use. This was possible because micro DBMS products operate on dedicated computers and because more computer power was available to process the user interface.

Today, DBMS products are rich and robust with graphical user interfaces such as Microsoft Windows.

The combination of microcomputers, the relational model, and vastly improved user interfaces enabled database technology to move from an organizational context to a personal-computing context. When this occurred, the number of sites that used database technology exploded. In 1980 there were about 10,000 sites suing DBMS products in the United States. Today there are well over 20 million such sites!

In the late 1980s, a new style of programming called object-oriented programming (OOP) began to be used, which has a substantially different orientation from that of traditional programming. In brief, the data structures processed with OOP are considerably more complex than those processed with traditional languages. These data structures also are difficult to store in existing relational DBMS products. As a consequence, a new category of DBMS products called object-oriented database systems is evolving to store and process OOP data structures.

For a variety of reasons, OOP has not yet been widely used for business information systems. First, it is difficult to use, and it is very expensive to develop OOP applications. Second, most organizations have millions or billions of bytes of data already organized in relational databases, and they are unwilling to bear the cost and risk required to convert those databases to an ODBMS format. Finally, most ODBMS have been developed to support engineering applications, and they do not have features and functions that are appropriate or readily adaptable to business information applications.

Consequently, for the foreseeable future, ODBMS are likely to occupy a niche in commercial information systems applications.

1990: Client-Server Database Applications

In the middle to late 1980s, end users began to connect their separated microcomputers using local area networks (LANs). These networks enabled computers to send data to one another at previously unimaginable rates. The first applications of this technology shared peripherals, such as large-capacity fast disks, expensive printers and plotters, and facilitated intercomputer communication via electronic mail. In time, however, end users wanted to share their databases as well, which led to the development of multi-user database applications on LANs.

The LAN-based multi-user architecture is considerably different from the multi-user architecture used on mainframe databases. With a mainframe, only one CPU is involved in database application processing,

but with LAN systems, many CPUs can be simultaneously involved. Because this situation was both advantageous (greater performance) and more problematic (coordinating) the actions of independent CPUs), it led to a new style of multi-user database processing called the client-server database architecture.

Not all database processing on a LAN is client-server processing. A simple, but less robust, mode of processing is called file-sharing architecture. A company like Treble Clef could most likely use either type since it is a small organization with modest processing requirements Larger workgroups, however, would require client-server processing.

2000: Databases Using Internet Technology

As shown in the Calvert Island Reservations. Center example, database technology is being used in conjunction with Internet technology to publish database data on the WWW. This same technology is used to publish applications over corporate and organizational intranets. Some experts believe that, in time, all database applications will be delivered using HTTP, XML, and related technologies – even personal databases that are "published" to a single person.

Because many database applications will use Internet technology to publish databases on organizational intranets and department LANs, it is incorrect to refer to this category of application as Internet databases. Rather, this text will employ the phrase databases using Internet technology instead.

This category stands on the leading edge of database technology today. XML in particular serves the needs of database applications exceptionally well, and it will likely be the basis of many new database products and services.

(10,636 symbols)

http://www.kean.edu/~rmelworm/3040-00/LuoDatabaseTimeLine.html

TEXT 20 SOFTWARE DEVELOPMENT LIFE CYCLE

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC Activities

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:

1. Communication

This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

2. Requirement Gathering

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given –

- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database or
- collecting answers from the questionnaires.

3. Feasibility Study

After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

4. System Analysis

At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

5. Software Design

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

6. Coding

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

7. Testing

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

8. Integration

Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

9. Implementation

This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

10. Operation and Maintenance

This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

11. Disposition

As time elapses, the software may decline on the performance front. It may go completely obsolete or may need intense upgradation. Hence a pressing need to eliminate a major portion of the system arises. This phase includes archiving data and required software components, closing down the system, planning disposition activity and terminating system at appropriate end-of-system time.

(3,903 symbols)

http://www.tutorialspoint.com/software_engineering/software_development_life_cycle.htm

TEXT 21 SOFTWARE DEVELOPMENT MODELS

The development models are the various processes or methodologies that are being selected for the development of the project depending on the project's aims and goals. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out.

The selection of model has very high impact on the testing that is carried out. It will define the what, where and when of our planned testing, influence regression testing and largely determines which test techniques to use.

There are various Software development models or methodologies. They are as follows:

1. Waterfall model
2. V model
3. Incremental model
4. RAD model
5. Iterative model
6. Spiral model
7. Agile model

Choosing right model for developing of the software product or application is very important. Based on the model the development and testing processes are carried out.

Different companies based on the software application or product, they select the type of development model whichever suits to their application. But these days in market the 'Agile Methodology' is the most used model. 'Waterfall Model' is the very old model. In 'Waterfall Model' testing starts only after the development is completed. Because of which there are many defects and failures which are reported at the end. So,the cost of fixing these issues are high. Hence, these days people are preferring 'Agile Model'. In 'Agile Model' after every sprint there is a demo-able feature to the customer. Hence customer can see the features whether they are satisfying their need or not.

'V-model' is also used by many of the companies in their product. 'V-model' is nothing but 'Verification' and 'Validation' model. In 'V-model' the developer's life cycle and tester's life cycle are mapped to each other. In this model testing is done side by side of the development.

Likewise ‘Incremental model’, ‘RAD model’, ‘Iterative model’ and ‘Spiral model’ are also used based on the requirement of the customer and need of the product.

1. Waterfall model

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model**.** It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In waterfall model phases do not overlap.

Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

2. V-model

V-model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development.

Advantages of V-model:

- Simple and easy to use.

- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

When to use the V-model:

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

High confidence of customer is required for choosing the V-Shaped model approach. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations.

3. Incremental model

In incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a“multi-waterfall” cycle. Cycles are divided up into smaller, more easily managed modules. Each module passes through the requirements, design, implementation and testing phases. A working version of software is produced during the first module, so you have working software early on during the software life cycle. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is achieved.

In the diagram above when we work incrementally we are adding piece by piece but expect that each piece is fully finished. Thus keep on adding the pieces until it’s complete. As in the image above a person has thought of the application. Then he started building it and in the first iteration the first module of the application or product is totally ready and can be demoed to the customers. Likewise in the second iteration the other module is ready and integrated with the first module. Similarly, in the third

iteration the whole product is ready and integrated. Hence, the product got ready step by step.

Advantages of Incremental model:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.

Disadvantages of Incremental model:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

When to use the Incremental model:

- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals.

4. RAD model

RAD model is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

Advantages of the RAD model:

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated codegeneration is very high.

When to use RAD model:

- RAD should be used when there is a need to create a system that can be modularized in 2–3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2–3 months).

5. Iterative model

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.

When we work iteratively we create rough product or product piece in one iteration, then review it and improve it in next iteration and so on until it's finished. As shown in the image above, in the first iteration the whole painting is sketched roughly, then in the second iteration colors are filled and in the third iteration finishing is done. Hence, in iterative model the whole product is developed step by step.

Advantages of Iterative model:

- In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
- In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.
- In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their

feedback, we are effectively asking them to imagine how the product will work.

- In iterative model less time is spent on documenting and more time is given for designing.

Disadvantages of Iterative model:

- Each phase of an iteration is rigid with no overlaps
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

6. Spiral model

The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

(9,666 symbols)

http://istqbexamcertification.com/what-are-the-software-development-models/

TEXT 22 WHAT IS EXTREME PROGRAMMING?

Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, courage, and respect. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.

Basic Extreme Programming

In Extreme Programming, every contributor to the project is an integral part of the "Whole Team". The team forms around a business representative called "the Customer", who sits with the team and works with them daily.

Extreme Programming teams use a simple form of planning and tracking to decide what should be done next and to predict when the project will be done. Focused on business value, the team produces the software in a series of small fully-integrated releases that pass all the tests the Customer has defined.

Extreme Programmers work together in pairs and as a group, with simple design and obsessively tested code, improving the design continually to keep it always just right for the current needs.

The Extreme Programming team keeps the system integrated and running all the time. The programmers write all production code in pairs, and all work together all the time. They code in a consistent style so that everyone can understand and improve all the code as needed.

Extreme Programming is about team responsibility for all code, for coding in a consistent pattern so that everyone can read everyone's code, about keeping the system running and integrated all the time.

The Extreme Programming team shares a common and simple picture of what the system looks like. Everyone works at a pace that can be sustained indefinitely.

Core Practices Whole Team

All the contributors to an XP project sit together, members of one team. This team must include a business representative – the "Customer" – who provides the requirements, sets the priorities, and steers the project. It's best if the Customer or one of her aides is a real end user who knows the domain and what is needed. The team will of course have programmers. The team may include testers, who help the Customer define the customer acceptance tests. Analysts may serve as helpers to the Customer, helping to define the requirements. There is commonly a coach,

who helps the team keep on track, and facilitates the process. There may be a manager, providing resources, handling external communication, coordinating activities. None of these roles is necessarily the exclusive property of just one individual: Everyone on an XP team contributes in any way that they can. The best teams have no specialists, only general contributors with special skills.

Planning Game

XP planning addresses two key questions in software development: predicting what will be accomplished by the due date, and determining what to do next. The emphasis is on steering the project – which is quite straightforward – rather than on exact prediction of what will be needed and how long it will take – which is quite difficult. There are two key planning steps in XP, addressing these two questions:

Release Planning is a practice where the Customer presents the desired features to the programmers, and the programmers estimate their difficulty. With the cost estimates in hand, and with knowledge of the importance of the features, the Customer lays out a plan for the project. Initial release plans are necessarily imprecise: neither the priorities nor the estimates are truly solid, and until the team begins to work, we won't know just how fast they will go. Even the first release plan is accurate enough for decision making, however, and XP teams revise the release plan regularly.

Iteration Planning is the practice whereby the team is given direction every couple of weeks. XP teams build software in two-week "iterations", delivering running useful software at the end of each iteration. During Iteration Planning, the Customer presents the features desired for the next two weeks. The programmers break them down into tasks, and estimate their cost (at a finer level of detail than in Release Planning). Based on the amount of work accomplished in the previous iteration, the team signs up for what will be undertaken in the current iteration.

These planning steps are very simple, yet they provide very good information and excellent steering control in the hands of the Customer. Every couple of weeks, the amount of progress is entirely visible. There is no "ninety percent done" in XP: a feature story was completed, or it was not. This focus on visibility results in a nice little paradox: on the one hand, with so much visibility, the Customer is in a position to cancel the project if progress is not sufficient. On the other hand, progress is so visible, and the ability to decide what will be done next is so complete, that XP projects tend to deliver more of what is needed, with less pressure and stress.

Customer Tests

As part of presenting each desired feature, the XP Customer defines one or more automated acceptance tests to show that the feature is working. The team builds these tests and uses them to prove to themselves, and to the customer, that the feature is implemented correctly. Automation is important because in the press of time, manual tests are skipped. That's like turning off your lights when the night gets darkest.

The best XP teams treat their customer tests the same way they do programmer tests: once the test runs, the team keeps it running correctly thereafter. This means that the system only improves, always notching forward, never backsliding.

Small Releases

XP teams practice small releases in two important ways:

First, the team releases running, tested software, delivering business value chosen by the Customer, every iteration. The Customer can use this software for any purpose, whether evaluation or even release to end users (highly recommended). The most important aspect is that the software is visible, and given to the customer, at the end of every iteration. This keeps everything open and tangible.

Second, XP teams release to their end users frequently as well. XP Web projects release as often as daily, in house projects monthly or more frequently. Even shrink-wrapped products are shipped as often as quarterly.

It may seem impossible to create good versions this often, but XP teams all over are doing it all the time.

Simple Design

XP teams build software to a simple but always adequate design. They start simple, and through programmer testing and design improvement, they keep it that way. An XP team keeps the design exactly suited for the current functionality of the system. There is no wasted motion, and the software is always ready for what's next.

Design in XP is not a one-time thing, or an up-front thing, it is an all-the-time thing. There are design steps in release planning and iteration planning, plus teams engage in quick design sessions and design revisions through refactoring, through the course of the entire project. In an incremental, iterative process like Extreme Programming, good design is essential. That's why there is so much focus on design throughout the course of the entire development.

Pair Programming

All production software in XP is built by two programmers, sitting side by side, at the same machine. This practice ensures that all production code is reviewed by at least one other programmer, and results in better design, better testing, and better code.

It may seem inefficient to have two programmers doing "one programmer's job", but the reverse is true. Research into pair programming shows that pairing produces better code in about the same time as programmers working singly. That's right: two heads really are better than one!

Some programmers object to pair programming without ever trying it. It does take some practice to do well, and you need to do it well for a few weeks to see the results. Ninety percent of programmers who learn pair programming prefer it, so we highly recommend it to all teams.

Pairing, in addition to providing better code and tests, also serves to communicate knowledge throughout the team. As pairs switch, everyone gets the benefits of everyone's specialized knowledge. Programmers learn, their skills improve, they become more valuable to the team and to the company. Pairing, even on its own outside of XP, is a big win for everyone.

Test-Driven Development

Extreme Programming is obsessed with feedback, and in software development, good feedback requires good testing. Top XP teams practice "test-driven development", working in very short cycles of adding a test, then making it work. Almost effortlessly, teams produce code with nearly 100 percent test coverage, which is a great step forward in most shops. (If your programmers are already doing even more sophisticated testing, more power to you. Keep it up, it can only help!)

It isn't enough to write tests: you have to run them. Here, too, Extreme Programming is extreme. These "programmer tests", or "unit tests" are all collected together, and every time any programmer releases any code to the repository (and pairs typically release twice a day or more), every single one of the programmer tests must run correctly. One hundred percent, all the time! This means that programmers get immediate feedback on how they're doing. Additionally, these tests provide invaluable support as the software design is improved.

Design Improvement (Refactoring)

Extreme Programming focuses on delivering business value in every iteration. To accomplish this over the course of the whole project,

the software must be well-designed. The alternative would be to slow down and ultimately get stuck. So XP uses a process of continuous design improvement called *Refactoring*, from the title of Martin Fowler's book, "Refactoring: Improving the Design of Existing Code".

The refactoring process focuses on removal of duplication (a sure sign of poor design), and on increasing the "cohesion" of the code, while lowering the "coupling". High cohesion and low coupling have been recognized as the hallmarks of well-designed code for at least thirty years. The result is that XP teams start with a good, simple design, and always have a good, simple design for the software. This lets them sustain their development speed, and in fact generally increase speed as the project goes forward.

Refactoring is, of course, strongly supported by comprehensive testing to be sure that as the design evolves, nothing is broken. Thus the customer tests and programmer tests are a critical enabling factor. The XP practices support each other: they are stronger together than separately.

Continuous Integration

Extreme Programming teams keep the system fully integrated at all times. We say that daily builds are for wimps: XP teams build multiple times per day. (One XP team of forty people builds at least eight or ten times per day!)

The benefit of this practice can be seen by thinking back on projects you may have heard about (or even been a part of) where the build process was weekly or less frequently, and usually led to "integration hell", where everything broke and no one knew why.

Infrequent integration leads to serious problems on a software project. First of all, although integration is critical to shipping good working code, the team is not practiced at it, and often it is delegated to people who are not familiar with the whole system. Second, infrequently integrated code is often – I would say usually – buggy code. Problems creep in at integration time that are not detected by any of the testing that takes place on an unintegrated system. Third, weak integration process leads to long code freezes. Code freezes mean that you have long time periods when the programmers could be working on important shippable features, but that those features must be held back. This weakens your position in the market, or with your end users.

Collective Code Ownership

On an Extreme Programming project, any pair of programmers can improve any code at any time. This means that all code gets the benefit of

many people's attention, which increases code quality and reduces defects. There is another important benefit as well: when code is owned by individuals, required features are often put in the wrong place, as one programmer discovers that he needs a feature somewhere in code that he does not own. The owner is too busy to do it, so the programmer puts the feature in his own code, where it does not belong. This leads to ugly, hard-to-maintain code, full of duplication and with low (bad) cohesion.

Collective ownership could be a problem if people worked blindly on code they did not understand. XP avoids these problems through two key techniques: the programmer tests catch mistakes, and pair programming means that the best way to work on unfamiliar code is to pair with the expert. In addition to ensuring good modifications when needed, this practice spreads knowledge throughout the team.

Coding Standard

XP teams follow a common coding standard, so that all the code in the system looks as if it was written by a single – very competent – individual. The specifics of the standard are not important: what is important is that all the code looks familiar, in support of collective ownership.

Metaphor

Extreme Programming teams develop a common vision of how the program works, which we call the "metaphor". At its best, the metaphor is a simple evocative description of how the program works, such as "this program works like a hive of bees, going out for pollen and bringing it back to the hive" as a description for an agent-based information retrieval system.

Sometimes a sufficiently poetic metaphor does not arise. In any case, with or without vivid imagery, XP teams use a common system of names to be sure that everyone understands how the system works and where to look to find the functionality you're looking for, or to find the right place to put the functionality you're about to add.

Sustainable Pace

Extreme Programming teams are in it for the long term. They work hard, and at a pace that can be sustained indefinitely. This means that they work overtime when it is effective, and that they normally work in such a way as to maximize productivity week in and week out. It's pretty well understood these days that death march projects are neither productive nor produce quality software. XP teams are in it to win, not to die.

Conclusion

Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.

(11,997 symbols)

http://ronjeffries.com/xprog/what-is-extreme-programming/

TEXT 23 SOFTWARE TESTING – METHODS

There are different methods that can be used for software testing. This chapter briefly describes the methods available.

Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

Advantages	Disadvantages
• Well suited and efficient for large code segments. • Code access is not required. • Clearly separates user's perspective from the developer's perspective through visibly defined roles. • Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	• Limited coverage, since only a selected number of test scenarios is actually performed. • Inefficient testing, due to the fact that the tester only has limited knowledge about an application. • Blind coverage, since the tester cannot target specific code segments or error-prone areas. • The test cases are difficult to design.

White-Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

Advantages	Disadvantages
• As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. • It helps in optimizing the code. • Extra lines of code can be removed which can bring in hidden defects. • Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	• Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased. • Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested. • It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.

Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages	Disadvantages
• Offers combined benefits of black-box and white-box testing wherever possible. • Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications. • Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling. • The test is done from the point of view of the user and not the designer.	• Since the access to source code is not available, the ability to go over the code and test coverage is limited. • The tests can be redundant if the software designer has already run a test case. • Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

A Comparison of Testing Methods

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.

Black-Box Testing	Grey-Box Testing	White-Box Testing
Testing is based on external expectations – Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

(4,448 symbols)

http://www.tutorialspoint.com/software_testing/software_testing_methods.htm

TEXT 24 MICROPROCESSORS: PAST, PRESENT AND FUTURE

Unlike many other technologies that fed our imaginations and then faded away, the computer has transformed our society. There can be little doubt that it will continue to do so for many decades to come. The engine driving this ongoing revolution is the microprocessor. These silicon chips have led to countless inventions, such as portable computers and fax machines, and have added intelligence to modern automobiles and wristwatches. Astonishingly, their performance has improved 25,000 times over since their invention. If the microprocessor continues to improve at its current rate, one cannot help but suggest that 25 years from now these chips will empower revolutionary software to compute wonderful things.

Smaller, Faster, Cheaper

Two inventions sparked the computer revolution. The first was the so-called stored program concept. Every computer system since the late 1940s

has adhered to this model, which prescribes a processor for crunching numbers and a memory for storing both data and programs. The advantage in such a system is that, because stored programs can be easily interchanged, the same hardware can perform a variety of tasks. Had computers not been given this flexibility, it is probable that they would not have met with such widespread use. Also, during the late 1940s, researchers invented the transistor. These silicon switches were much smaller than the vacuum tubes used in early circuitry. As such, they enabled workers to create smaller – and faster-electronics.

More than a decade passed before the stored program design and transistors were brought together in the same machine, and it was not until 1971 that the most significant pairing – the Intel 4004 –came about. This processor was the first to be built on a single silicon chip, which was no larger than a child's fingernail. Because of its tiny size, it was dubbed a microprocessor. And because it was a single chip, the Intel 4004 was the first processor that could be made inexpensively in bulk.

The method manufacturers have used to mass-produce microprocessors since then is much like baking a pizza: the dough, in this case silicon, starts thin and round. Chemical toppings are added, and the assembly goes into an oven. Heat transforms the toppings into transistors, conductors and insulators. Not surprisingly, the process – which is repeated perhaps 20 times – is considerably more demanding than baking a pizza. One dust particle can damage the tiny transistors. So, too, vibrations from a passing truck can throw the ingredients out of alignment, ruining the end product. But provided that does not happen, the resulting wafer is divided into individual pieces, called chips, and served to customers.

Although this basic recipe is still followed, the production line has made ever cheaper, faster chips over time by churning out larger wafers and smaller transistors. This trend reveals an important principle of microprocessor economics: the more chips made per wafer, the less expensive they are. Larger chips are faster than smaller ones because they can hold more transistors. The recent Intel P6, for example, contains 5.5 million transistors and is much larger than the Intel 4004, which had a mere 2,300 transistors. But larger chips are also more likely to contain flaws. Balancing cost and performance, then, is a significant part of the art of chip design.

Most recently, microprocessors have become more powerful, thanks to a change in the design approach. Following the lead of researchers at universities and laboratories across the U.S., commercial chip designers

now take a quantitative approach to computer architecture. Careful experiments precede hardware development, and engineers use sensible metrics to judge their success. Computer companies acted in concert to adopt this design strategy during the 1980s, and as a result, the rate of improvement in microprocessor technology has risen from 3 5 percent a year only a decade ago to its current high of approximately 55 percent a year, or almost 4 percent each month. Processors are now three times faster than had been predicted in the early 1980s; it is as if our wish was granted, and we now have machines from the year 2000.

Pipelined, Superscalar and Parallel

In addition to progress made on the -L production line and in silicon technology, microprocessors have benefited from recent gains on the drawing board. These breakthroughs will undoubtedly lead to further advancements in the near future. One key technique is called pipelining. Anyone who has done laundry has intuitively used this tactic. The nonpipelined approach is as follows: place a load of dirty clothes in the washer. When the washer is done, place the wet load into the dryer. When the dryer is finished, fold the clothes. After the clothes are put away, start all over again. If it takes an hour to do one load this way, 20 loads take 20 hours.

The pipelined approach is much quicker. As soon as the first load is in the dryer, the second dirty load goes into the washer, and so on. All the stages operate concurrently. The pipelining paradox is that it takes the same amount of time to dean a single dirty sock by either method. Yet pipelining is faster in that more loads are finished per hour. In fact, assuming that each stage takes the same amount of time, the time saved by pipelining is proportional to the number of stages involved. In our example, pipelined laundry has four stages, so it would be nearly four times faster than nonpipelined laundry. Twenty loads would take roughly five hours.

Similarly, pipelining makes for much faster microprocessors. Chip designers pipeline the instructions, or low-level commands, given to the hardware. The first pipelined microprocessors used a five-stage pipeline. (The number of stages completed each second is given by the so-called dock rate. A personal computer with a 100-megahertz dock then executes 100 million stages per second.) Because the speedup from pipelining equals the number of stages, recent microprocessors have adopted eight or more stage pipelines. One 1995 microprocessor uses this deeper pipeline to achieve a 300-megahertz clock rate. As machines head toward the next

century, we can expect pipelines having even more stages and higher dock rates.

Also in the interest of making faster chips, designers have begun to include more hardware to process more tasks at each stage of a pipeline. The buzzword "superscalar" is commonly used to describe this approach. A superscalar laundromat, for example, would use a professional machine that could, say, wash three loads at once. Modern superscalar microprocessors try to perform anywhere from three to six instructions in each stage. Hence, a 250-megahertz, four-way superscalar microprocessor can execute a billion instructions per second. A 21st-century micro-processor may well launch up to dozens of instructions in each stage.

Despite such potential, improvements in processing chips are ineffectual unless they are matched by similar gains in memory chips. Since random-access memory (RAM) on a chip became widely available in the mid-1970s, its capacity has grown fourfold every three years. But memory speed has not increased at anywhere near this rate. The gap between the top speed of processors and the top speed of memories is widening.

One popular aid is to place a small memory, called a cache, right on the microprocessor itself. The cache holds those segments of a program that are most frequently used and thereby allows the processor to avoid calling on external memory chips much of the time. Some newer chips actually dedicate as many transistors to the cache as they do to the processor itself. Future microprocessors will allot even more resources to the cache to better bridge the speed gap.

The Holy Grail of computer design is an approach called parallel processing, which delivers all the benefits of a single fast processor by engaging many inexpensive ones at the same time. In our analogy, we would go to a laundromat and use 20 washers and 20 dryers to do 20 loads simultaneously. Clearly, parallel processing is an expensive solution for a small workload. And writing a program that can use 20 processors at once is much harder than distributing laundry to 20 washers.

Indeed, the program must specify which instructions can be launched by which processor at what time. Superscalar processing bears similarities to parallel processing, and it is more popular because the hardware automatically finds instructions that launch at the same time. But its potential processing power is not as large. If it were not so difficult to write the necessary programs, parallel processors could be made as powerful as one could afford. For the past 25 years, computer scientists have predicted

that the programming problems will be overcome. In fact, parallel processing is practical for only a few classes of programs today. In reviewing old articles, I have seen fantastic predictions of what computers would be like in 1995. Many stated that optics would replace electronics; computers would be built entirely from biological materials; the stored program concept would be discarded. These descriptions demonstrate that it is impossible to foresee what inventions will prove commercially viable and go on to revolutionize the computer industry. In my career, only three new technologies have prevailed: microprocessors, random-access memory and optical fibers. And their impact has yet to wane, decades after their debut.

IRAMs and Picoprocessors

Pipelining, superscalar organization and caches will continue to play major roles in the advancement of microprocessor technology, and if hopes are realized, parallel processing will join them. What will be startling is that microprocessors will probably exist in everything from light switches to pieces of paper. And the range of applications these extraordinary devices will support, from voice recognition to virtual reality, will very likely be astounding.

Today microprocessors and memories are made on distinct manufacturing lines, but it need not be so. Perhaps in the near future, processors and memory will be merged onto a single chip, just as the microprocessor first merged the separate components of a processor onto a single chip. To narrow the processor-memory performance gap, to take advantage of parallel processing, to amortize the costs of the line and simply to make full use of the phenomenal number of transistors that can be placed on a single chip, I predict that the high-end microprocessor of 2020 will be an entire computer.

Let’s call it an IRAM, standing for intelligent random-access memory, since most of the transistors on this merged chip will be devoted to memory. Whereas current microprocessors rely on hundreds of wires to connect to external memory chips, IRAMs will need no more than computer network connections and a power plug. All input-output devices will be linked to them via networks. If they need more memory, they will get more processing power as well, and vice versa – an arrangement that will keep the memory capacity and processor speed in balance. IRAMs are also the ideal building block for parallel processing. And because they would require so few external connections, these chips could be extraordinarily small. We may well see cheap “picoprocessors” that are

smaller than the ancient Intel 4004. If parallel processing succeeds, this sea of transistors could also be used by multiple processors on a single chip, giving us a micromultiprocessor.

Today's microprocessors are almost 100,000 times faster than their Neanderthal ancestors of the 1950s, and when inflation is considered, they cost 1,000 times less. These extraordinary facts explain why computing plays such a large role in our world now. Looking ahead, microprocessor performance will easily keep doubling every 18 months through the turn of the century. After that, it is hard to bet against a curve that has outstripped all expectations. But it is plausible that we will see improvements in the next 25 years at least as large as those seen in the past 50. This estimate means that one desktop computer in the near future will be as powerful as all the computers in Silicon Valley today.

(9,927 symbols)

http://www.scientificamerican.com/

TEXT 25 WEB DEVELOPMENT

Web development refers to building, creating, and maintaining websites. It includes aspects such as web design, web publishing, web programming, and database management.

While the terms "web developer" and "web designer" are often used synonymously, they do not mean the same thing. Technically, a web designer only designs website interfaces using HTML and CSS. A web developer may be involved in designing a website, but may also write web scripts in languages such as PHP and ASP. Additionally, a web developer may help maintain and update a database used by a dynamic website.

Web development includes many types of web content creation. Some examples include hand coding web pages in a text editor, building a website in a program like Dreamweaver, and updating a blog via a blogging website. In recent years, content management systems like WordPress, Drupal, and Joomla have also become popular means of web development. These tools make it easy for anyone to create and edit their own website using a web-based interface.

While there are several methods of creating websites, there is often a trade-off between simplicity and customization. Therefore, most large businesses do not use content management systems, but instead have a dedicated Web development team that designs and maintains the company's website(s). Small organizations and individuals are more likely

to choose a solution like WordPress that provides a basic website template and simplified editing tools.

NOTE: JavaScript programming is a type of web development that is generally not considered part of web design. However, a web designer may reference JavaScript libraries like jQuery to incorporate dynamic elements into a site's design.

Web design is the process of creating websites. It encompasses several different aspects, including webpage layout, content production, and graphic design. While the terms web design and web development are often used interchangeably, web design is technically a subset of the broader category of web development.

Websites are created using a markup language called HTML. Web designers build webpages using HTML tags that define the content and metadata of each page. The layout and appearance of the elements within a webpage are typically defined using CSS, or cascading style sheets. Therefore, most websites include a combination of HTML and CSS that defines how each page will appear in a browser.

Some web designers prefer to hand code pages (typing HTML and CSS from scratch), while others use a "WYSIWYG" editor like Adobe Dreamweaver. This type of editor provides a visual interface for designing the webpage layout and the software automatically generates the corresponding HTML and CSS code. Another popular way to design websites is with a content management system like WordPress or Joomla. These services provide different website templates that can be used as a starting point for a new website. Webmasters can then add content and customize the layout using a web-based interface.

While HTML and CSS are used to design the look and feel of a website, images must be created separately. Therefore, graphic design may overlap with web design, since graphic designers often create images for use on the Web. Some graphics programs like Adobe Photoshop even include a "Save for Web…" option that provides an easy way to export images in a format optimized for web publishing.

Web publishing, or "online publishing," is the process of publishing content on the Internet. It includes creating and uploading websites, updating webpages, and posting blogs online. The published content may include text, images, videos, and other types of media.

In order to publish content on the web, you need three things: 1) web development software, 2) an Internet connection, and 3) a web server. The software may be a professional web design program like

Dreamweaver or a simple web-based interface like WordPress. The Internet connection serves as the medium for uploading the content to the web server. Large sites may use a dedicated web host, but many smaller sites often reside on shared servers, which host multiple websites. Most blogs are published on public web servers through a free service like Blogger.

Since web publishing doesn't require physical materials such as paper and ink, it costs almost nothing to publish content on the web. Therefore, anyone with the three requirements above can be a web publisher. Additionally, the audience is limitless since content posted on the web can be viewed by anyone in the world with an Internet connection. These advantages of web publishing have led to a new era of personal publishing that was not possible before.

NOTE: Posting updates on social networking websites like Facebook and Twitter is generally not considered web publishing. Instead, web publishing generally refers to uploading content to unique websites.

What is the difference between a web designer and a web developer? In the early days of the web, the answer to that question was simple: designers design and developers code.

Today that question requires a little more nuance – you'd be hard pressed to find a web designer who didn't know at least a little HTML and CSS, and you won't have to look far for a front-end web developer who can whip up a storyboard.

If you're strictly speaking about the general concepts of web design vs. web development however, the distinction is a little more clear. Let's take a look at these two concepts and the roles they play in building the websites and apps we know and love.

What is web design?

Web design governs everything involved with the **visual aesthetics** and **usability** of a website – color scheme, layout, information flow, and everything else related to the visual aspects of the UI/UX (user interface and user experience). Some common skills and tools that distinguish the web designer from the web developer are:

- Adobe Creative Suite (Photoshop, Illustrator) or other design software
- Graphic design
- Logo design
- Layout/format

- Placing call-to-action buttons
- Branding
- Wireframes, mock-ups, and storyboards
- Color palettes
- Typography

Web design is concerned with what the user actually sees on their computer screen or mobile device, and less so about the mechanisms beneath the surface that make it all work. Through the use of color, images, typography and layout, they bring a digital experience to life.

That said, many web designers are also familiar with HTML, CSS, and JavaScript – it helps to be able to create living mock-ups of a web app when trying to pitch an idea to the team or fine-tune the UI/UX of an app. Web designers also often work with templating services like WordPress or Joomla!, which allow you to create websites using themes and widgets without writing a single line of code.

What is web development?

Web development governs all the code that makes a website tick. It can be split into two categories – front-end and back-end. The front-end or client-side of an application is the code responsible for determining how the website will actually display the designs mocked up by a designer. The back-end or server-side of an application is responsible for managing data within the database and serving that data to the front-end to be displayed. As you may have guessed, it's the front-end developer's job that tends to share the most overlap with the web designer. Some common skills and tools traditionally viewed as unique to the front-end developer are listed below:

- HTML/CSS/JavaScript
- CSS preprocessors (i.e., LESS or Sass)
- Frameworks (i.e., AngularJS, ReactJS, Ember)
- Libraries (i.e., jQuery)
- Git and GitHub

Front-end web developers don't usually create mock-ups, select typography, or pick color palettes – these are usually provided by the designer. It's the developer's job to bring those mock-ups to life. That said, understanding what the designer wants requires some knowledge of best practices in UI/UX design, so that the developer is able to choose the right technology to deliver the desired look and feel and experience in the final product.

Meet the "unicorn"

What started out as a joke in the industry – the designer/developer hybrid who can do it all – is now a viable endgame for both web designers and front-end developers, thanks to the increase in availability of educational resources across the web. Those developers/designers who have a good grasp of skills across both sides of the spectrum are highly sought after in the industry. The "unicorn" can take your project from the conceptual stage of visual mock-ups and storyboards, and carry it through front-end development all by themselves. Not that you'd want them to; the real value of developers who design and designers who develop is their ability to speak each other's languages. This leads not only to better communication on the team and a smoother workflow, it means you'll land on the best solution possible. As a general rule, feel free to rely on the "unicorn" for small projects, where it's feasible for one or two people to handle both the back and front-ends of an application. For larger projects, even if you do manage to hire a few "unicorns," more clearly defined roles are required.

(7,462 symbols)
http://techterms.com/definition/web_development

TEXT 26 DBMS

A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data.

A DBMS makes it possible for end users to create, read, update and delete data in a database. The DBMS essentially serves as an interface between the database and end users or application programs, ensuring that data is consistently organized and remains easily accessible.

The DBMS manages three important things: the data, the database engine that allows data to be accessed, locked and modified – and the database schema, which defines the database's logical structure. These three foundational elements help provide concurrency, security, data integrity and uniform administration procedures. Typical database administration tasks supported by the DBMS include change management, performance monitoring/tuning and backup and recovery. Many database management systems are also responsible for automated rollbacks, restarts and recovery as well as the logging and auditing of activity.

The DBMS is perhaps most useful for providing a centralized view of data that can be accessed by multiple users, from multiple locations, in a controlled manner. A DBMS can limit what data the end user sees, as well as how that end user can view the data, providing many views of a single database schema. End users and software programs are free from having to understand where the data is physically located or on what type of storage media it resides because the DBMS handles all requests.

The DBMS can offer both logical and physical data independence. That means it can protect users and applications from needing to know where data is stored or having to be concerned about changes to the physical structure of data (storage and hardware). As long as programs use the application programming interface (API) for the database that is provided by the DBMS, developers won't have to modify programs just because changes have been made to the database.

With relational DBMSs (RDBMSs), this API is SQL, a standard programming language for defining, protecting and accessing data in a RDBMS.

Popular types of DBMSes

Popular database models and their management systems include:

Relational database management system (RDMS) – adaptable to most use cases, but RDBMS Tier-1 products can be quite expensive.

NoSQL DBMS - well-suited for loosely defined data structures that may evolve over time.

In-memory database management system (IMDBMS) – provides faster response times and better performance.

Columnar database management system (CDBMS) – well-suited for data warehouses that have a large number of similar data items.

Cloud-based data management system - the cloud service provider is responsible for providing and maintaining the DBMS.

Advantages of a DBMS

Using a DBMS to store and manage data comes with advantages, but also overhead. One of the biggest advantages of using a DBMS is that it lets end users and application programmers access and use the same data while managing data integrity. Data is better protected and maintained when it can be shared using a DBMS instead of creating new iterations of the same data stored in new files for every new application. The DBMS provides a central store of data that can be accessed by multiple users in a controlled manner.

DBMS provides:

- Data abstraction and independence
- Data security
- A locking mechanism for concurrent access
- An efficient handler to balance the needs of multiple applications using the same data
- The ability to swiftly recover from crashes and errors, including restartability and recoverability
- Robust data integrity capabilities
- Logging and auditing of activity
- Simple access using a standard application programming interface (API)
- Uniform administration procedures for data

Another advantage of a DBMS is that it can be used to impose a logical, structured organization on the data. A DBMS delivers economy of scale for processing large amounts of data because it is optimized for such operations.

A DBMS can also provide many views of a single database schema. A view defines what data the user sees and how that user sees the data. The DBMS provides a level of abstraction between the conceptual schema that defines the logical structure of the database and the physical schema that describes the files, indexes and other physical mechanisms used by the database. When a DBMS is used, systems can be modified much more easily when business requirements change. New categories of data can be added to the database without disrupting the existing system and applications can be insulated from how data is structured and stored.

Of course, a DBMS must perform additional work to provide these advantages, thereby bringing with it the overhead. A DBMS will use more memory and CPU than a simple file storage system. And, of course, different types of DBMSes will require different types and levels of system resources.

A NoSQL DBMS (Not only SQL database management system) is system software designed to create and manage NoSQL databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data.

The world in which relational database management systems (RDBMSes) are the norm is fast disappearing. Although relational database systems remain important, multiple new types of database systems – of which the fastest growing is NoSQL – are being designed

and implemented to meet new types of business needs. Understanding the types of NoSQL DBMSes that are available, as well as how and when they can be beneficial, is an important requirement for modern application development, especially mobile application development.

(4,726 symbols)

http://searchsqlserver.techtarget.com/definition/database-management-system

TEXT 27 1C:ENTERPRISE

1C:Enterprise is a universal cloud and on-premise system of programs for automating a company's financial and wider operational activities. 1C:Enterprise has the breadth of capability to address the diverse needs of today's business. This is achieved through "configurability" – the ability to customize the system based on the specific needs of companies and their business processes.

1C:Enterprise is more than just a solution automating fixed business rules. Rather it is a suite of software tools employed by developers and users. The system can be logically divided into two major components that are closely interrelated: an application and the platform on which the application runs.

1C:Enterprise Platform offers the following advantages:

- Drastically reduces technological complexity, ergonomics, and performance issues of enterprise software systems.
- Allows power users to implement specific business processes.
- Speeds up and standardizes business application development, customization, and support.
- Provides complete openness of UI and solution code, which allows better understanding and modification of business processes.
- Is ready for integration with existing 1C applications and third-party systems.
- Supports web-services, ODBC, COM, and so on.
- Supports your preferred architecture: Windows/Linux, MS SQL, PostgreSQL, IBM DB2, and Oracle DB.
- Includes on-premise or managed hosting web-based delivery, as well as Web, tablet, or Windows client.

The 1C:Enterprise System of Programs

The 1C:Enterprise application system is daily used by several million users in business and government to automate operations, accounting,

finance, HR, and management activities. 1C Company provides an array of vertical solutions for manufacturing, distribution, and service businesses. With its innovative technological platform and the array of applied solutions, 1C Company has achieved wide popularity for its openness, speed of modification and software updates. 1C:Enterprise is a very flexible and scalable platform meeting the needs of companies ranging in size from a single user to hundreds of users.

Configurability

A major feature of the 1C:Enterprise system is its configurability.

The 1C:Enterprise system itself is a set of mechanisms designed to manipulate various types of objects in a subject area. A set of objects, data array structures and information processing algorithms for the assigned task are defined by a specific configuration. Joined with the configuration, the 1C:Enterprise system functions as a ready-to-use software product customized for particular enterprise types and task classes.

The configuration is created and supported by standard system tools. A configuration is usually supplied as standard for a particular area of application, but it may be modified or extended by the user or developed from scratch. The 1C:Enterprise system supports standard configurations using standard tools.

Functioning of the system

Functioning of the system involves two processes: development (description of a subject-area model by system tools) and execution (processing of subject-area data).

The development stage includes:

Structuring the processed information
Creating forms for source data entry and data list display
Storing the entered and derived information
Generating reports and data processors
Creating command interfaces for various user groups
Creating a user list
Assigning rights to users

Development results in software (configuration) representing a model.

The designing mode permits the user to create new configurations, edit the existing ones and compare or merge several configurations.

At the development stage, the system uses universal concepts or objects such as Document, Document Journal, Catalog, Attribute, Form, Register and others. A set of these concepts defines the concept of the system. In its turn, the configuration process is broken down into several

components (this is an arbitrary division) that define the order in which volumes of description are written and assigned. These are "visual" configuration (creation of the configuration structure, forms for dialog boxes and output documents, user-data interaction mechanism or interface and access rights of various user groups to various types of information) and generation of programs in the 1C:Enterprise script for processing input and output data.

The actual concepts of objects and standard operations for processing them are defined at the system level. Configuration tools can be used to describe the structure of information included in the objects and algorithms that describe the specifics of their processing to account for their various accounting features.

The information structure is designed at the level of the processable subject-area objects specified in the system (constants, catalogs, documents, registers, enumerations, etc.).

As it runs, the system works with specific concepts described at the configuration stage (product and organization catalogs, bills, invoices, etc.).

When the user works in the 1C:Enterprise mode, information is processed both with standard system tools and using algorithms created at the configuration stage.

Basic concepts of the system

This section discusses the basic concepts used by the 1C:Enterprise system. It is useful to those who are not yet familiar with the 1C:Enterprise system.

The description of various mechanisms is illustrated with examples. You can encounter unfamiliar terms and concepts in the description. Keep reading – the meaning of the terms used will become clear as you do so; if you want more detailed information, you can always refer to the corresponding chapters of this guide.

Configuration Concept

The basic concept is that of configuration.

In the 1C:Enterprise system, a configuration means a set of interrelated components:

subsystems

accounting data structures and data input, selection and print forms

a set of mechanisms for totals accounting and register records of accounting data

a set of various reports and data processors

command interface

a set of roles or access rights

a set of common procedures and functions (application module, managed application module, external connection module, session module, and common modules), spreadsheet templates, etc.

auxiliary objects: functional options and their parameters

settings storages

Web tools (Web-services, WS references)

various auxiliary information (pictures, templates, styles, etc.)

In fact, a configuration structure is a subject area model. A configuration is created using the Designer. The resulting configuration is used by the 1C:Enterprise system to create a software environment suitable for the accomplishment of the necessary accounting tasks.

Roles in the 1C:Enterprise system define whether users can work with information processed in the system. A set of privileges granted to the user is generally defined by the scope of the user's duties.

Assignment of roles to the user accomplishes two things:

On the one hand, it limits the number of users having access to sensitive information that is always a part of any accounting system.

On the other hand, prohibition of certain operations (primarily data deletion and editing) helps prevent a possible loss of information.

All components of a configuration are closely interrelated and generally require consistency in making changes (this applies especially to user rights).

Thus, roles can be assigned only for existing configuration objects (specific documents, journals, catalogs or reports). Insertion of an object into the configuration structure must be accompanied by appropriate role changes.

(6,317 symbols)

http://1c-dn.com/

TEXT 28 ORACLE DATABASE ARCHITECTURE

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance.

A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.

The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

Overview of Oracle Grid Architecture.

Grid computing is a new IT architecture that produces more resilient and lower cost enterprise information systems. With grid computing, groups of independent, modular hardware and software components can be connected and rejoined on demand to meet the changing needs of businesses.

The grid style of computing aims to solve some common problems with enterprise IT: the problem of application silos that lead to under utilized, dedicated hardware resources, the problem of monolithic, unwieldy systems that are expensive to maintain and difficult to change, and the problem of fragmented and disintegrated information that cannot be fully exploited by the enterprise as a whole.

Benefits of Grid Computing

Compared to other models of computing, IT systems designed and implemented in the grid style deliver higher quality of service, lower cost, and greater flexibility. Higher quality of service results from having no single point of failure, a robust security infrastructure, and centralized, policy-driven management. Lower costs derive from increasing the utilization of resources and dramatically reducing management and maintenance costs. Rather than dedicating a stack of software and hardware to a specific task, all resources are pooled and allocated on demand, thus eliminating under utilized capacity and redundant capabilities. Grid computing also enables the use of smaller individual hardware components, thus reducing the cost of each individual component and providing more flexibility to devote resources in accordance with changing needs.

Grid Computing Defined

The grid style of computing treats collections of similar IT resources holistically as a single pool, while exploiting the distinct nature of individual resources within the pool. To address simultaneously the problems of monolithic systems and fragmented resources, grid computing achieves a balance between the benefits of holistic resource management and flexible independent resource control. IT resources managed in a grid include:

- Infrastructure: the hardware and software that create a data storage and program execution environment
- Applications: the program logic and flow that define specific business processes
- Information: the meanings inherent in all different types of data used to conduct business

Core Tenets of Grid Computing**.**

Two core tenets uniquely distinguish grid computing from other styles of computing, such as mainframe, client-server, or multi-tier: virtualization and provisioning.

- With virtualization, individual resources (e.g. computers, disks, application components and information sources) are pooled together by type then made available to consumers (e.g. people or software programs) through an abstraction. Virtualization means breaking hard-coded connections between providers and consumers of resources, and preparing a resource to serve a particular need without the consumer caring how that is accomplished.
- With provisioning, when consumers request resources through a virtualization layer, behind the scenes a specific resource is identified to fulfill the request and then it is allocated to the consumer. Provisioning as part of grid computing means that the system determines how to meet the specific need of the consumer, while optimizing operation of the system as a whole.

The specific ways in which information, application or infrastructure resources are virtualized and provisioned are specific to the type of resource, but the concepts apply universally. Similarly, the specific benefits derived from grid computing are particular to each type of resource, but all share the characteristics of better quality, lower costs and increased flexibility.

Infrastructure Grid

Infrastructure grid resources include hardware resources such as storage, processors, memory, and networks as well as software designed to manage this hardware, such as databases, storage management, system management, application servers, and operating systems.

Virtualization and provisioning of infrastructure resources mean pooling resources together and allocating to the appropriate consumers based on policies. For example, one policy might be to dedicate enough processing power to a web server that it can always provide sub-second response time. That rule could be fulfilled in different ways by the provisioning software in order to balance the requests of all consumers.

Treating infrastructure resources as a single pool and allocating those resources on demand saves money by eliminating under utilized capacity and redundant capabilities. Managing hardware and software resources holistically reduces the cost of labor and the opportunity for human error.

Spreading computing capacity among many different computers and spreading storage capacity across multiple disks and disk groups removes single points of failure so that if any individual component fails, the system as a whole remains available. Furthermore, grid computing affords the option to use smaller individual hardware components, such as blade servers and low cost storage, which enables incremental scaling and reduces the cost of each individual component, thereby giving companies more flexibility and lower cost.

Infrastructure is the dimension of grid computing that is most familiar and easy to understand, but the same concepts apply to applications and information.

(5,574 symbols)

http://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm

TEXT 29 FLIGHT INSTRUMENTS

Flight instruments are the instruments in the cockpit of an aircraft that provide the pilot with information about the flight situation of that aircraft, such as altitude, airspeed and direction. They improve safety by allowing the pilot to fly the aircraft in level flight, and make turns, without a reference outside the aircraft such as the horizon. Visual flight rules (VFR) require an airspeed indicator, an altimeter, and a compass or other suitable magnetic direction indicator. Instrument flight rules (IFR) additionally require a gyroscopic pitch-bank (artificial horizon), direction

(directional gyro) and rate of turn indicator, plus a slip-skid indicator, adjustable altimeter, and a clock. Flight into Instrument meteorological conditions (IMC) require radio navigation instruments for precise takeoffs and landings.

The term is sometimes used loosely as a synonym for cockpit instruments as a whole, in which context it can include engine instruments, navigational and communication equipment. Many modern aircraft have electronic flight instrument systems.

Most regulated aircraft have these flight instruments as dictated by the US Code of Federal Regulations, Title 14, Part 91. They are grouped according to pitot-static system, compass systems, and gyroscopic instruments.

Pitot-Static Systems

Altimeter

The altimeter shows the aircraft's altitude above sea-level by measuring the difference between the pressure in a stack of aneroid capsules inside the altimeter and the atmospheric pressure obtained through the static system. It is adjustable for local barometric pressure which must be set correctly to obtain accurate altitude readings. As the aircraft ascends, the capsules expand and the static pressure drops, causing the altimeter to indicate a higher altitude. The opposite effect occurs when descending. With the advancement in aviation and increased altitude ceiling the altimeter dial had to be altered for use both at higher and lower altitudes. Hence when the needles were indicating lower altitudes i.e. the first 360 degree operation of the pointers was delineated by the appearance of a small window with oblique lines warning the pilot that he/she is nearer to the ground. This modification was introduced in the early sixties after the recurrence of air accidents caused by the confusion in the pilot's mind. At higher altitudes the window will disappear.

Airspeed indicator

The airspeed indicator shows the aircraft's speed (usually in knots) relative to the surrounding air. It works by measuring the ram-air pressure in the aircraft's Pitot tube relative to the ambient static pressure. The Indicated airspeed (IAS) must be corrected for nonstandard pressure and temperature in order to obtain the True airspeed (TAS). The instrument is color coded to indicate important airspeeds such as the stall speed, never-exceed airspeed, or safe flap operation speeds.

Vertical speed indicator

The VSI (also sometimes called a variometer, or rate of climb indicator) senses changing air pressure, and displays that information to the pilot as a rate of climb or descent in feet per minute, meters per second or knots.

Compass Systems

Magnetic compass

The compass shows the aircraft's heading relative to magnetic north. Errors include Variation, or the difference between magnetic and true direction, and Deviation, caused by the electrical wiring in the aircraft, which requires a Compass Correction Card. Additionally, the compass is subject to Dip Errors. While reliable in steady level flight it can give confusing indications when turning, climbing, descending, or accelerating due to the inclination of the Earth's magnetic field. For this reason, the heading indicator is also used for aircraft operation, but periodically calibrated against the compass.

Gyroscopic Systems

Attitude Indicator

The attitude indicator (also known as an *artificial horizon*) shows the aircraft's relation to the horizon. From this the pilot can tell whether the wings are level (roll) and if the aircraft nose is pointing above or below the horizon (pitch). This is a primary instrument for instrument flight and is also useful in conditions of poor visibility. Pilots are trained to use other instruments in combination should this instrument or its power fail.

Schempp-Hirth Janus-C glider Instrument panel equipped for "cloud flying". The turn and bank indicator is top centre. The heading indicator is replaced by a GPS-driven computer with wind and glide data, driving two electronic variometer displays to the right.

Heading Indicator

The heading indicator (also known as the directional gyro, or DG) displays the aircraft's heading with respect to magnetic north when set with a compass. Bearing friction causes drift errors from precession, which must be periodically corrected by calibrating the instrument to the magnetic compass. In many advanced aircraft (including almost all jet aircraft), the heading indicator is replaced by a horizontal situation indicator (HSI) which provides the same heading information, but also assists with navigation.

Turn Indicator

These include the Turn-and-Slip Indicator and the Turn Coordinator, which indicate rotation about the longitudinal axis. They include an inclinometer to indicate if the aircraft is in Coordinated flight, or in a Slip or Skid. Additional marks indicate a Standard rate turn.

Flight Director Systems

These include the Horizontal Situation Indicator (HSI) and Attitude Director Indicator (ADI). The HSI combines the magnetic compass with navigation signals and a Glide slope. The navigation information comes from a VOR/Localizer, or GPS. The ADI is an Attitude Indicator with computer-driven steering bars, a task reliever during instrument flight.

Navigational Systems

Very-High Frequency Omnidirectional Range (VOR)

The VOR indicator instrument includes a Course deviation indicator (CDI), Omnibearing Selector (OBS), TO/FROM indicator, and Flags. The CDI shows an aircraft's lateral position in relation to a selected radial track. It is used for orientation, tracking to or from a station, and course interception.

Nondirectional Radio Beacon (NDB)

The Automatic direction finder (ADF) indicator instrument can be a fixed-card, movable card, or a Radio magnetic indicator (RMI). An RMI is remotely coupled to a gyrocompass so that it automatically rotates the azimuth card to represent aircraft heading. While simple ADF displays may have only one needle, a typical RMI has two, coupled to different ADF receivers, allowing for position fixing using one instrument.

Layout

Six basic instruments in a light twin-engine airplane arranged in a "basic-T". From top left: airspeed indicator, attitude indicator, altimeter, turn coordinator, heading indicator, and vertical speed indicator**.**

Most aircraft are equipped with a standard set of flight instruments which give the pilot information about the aircraft's attitude, airspeed, and altitude.

T arrangement

Most US aircraft built since the 1940s have flight instruments arranged in a standardized pattern called the "T" arrangement. The attitude indicator is in the top center, airspeed to the left, altimeter to the right and heading indicator under the attitude indicator. The other two, turn-coordinator and vertical-speed, are usually found under the airspeed and altimeter, but are given more latitude in placement. The magnetic compass

will be above the instrument panel, often on the windscreen centerpost. In newer aircraft with glass cockpit instruments the layout of the displays conform to the basic T arrangement.

Early history

In 1929, Jimmy Doolittle became the first pilot to take off, fly and land an airplane using instruments alone, without a view outside the cockpit. In 1937, the British Royal Air Force (RAF) chose a set of six essential flight instruments which would remain the standard panel used for flying in instrument meteorological conditions (IMC) for the next 20 years. They were:

- altimeter (feet)
- airspeed indicator (knots)
- turn and bank indicator (turn direction and coordination)
- vertical speed indicator (feet per minute)
- artificial horizon (attitude indication)
- directional gyro / heading indicator (degrees)

This panel arrangement was incorporated into all RAF aircraft built to official specification from 1938, such as the Miles Master, Hawker Hurricane, Supermarine Spitfire, and 4-engined Avro Lancaster and Handley Page Halifax heavy bombers, but not the earlier light single-engined Tiger Moth trainer, and minimized the type-conversion difficulties associated with blind flying, since a pilot trained on one aircraft could quickly become accustomed to any other if the instruments were identical.

This basic six set, also known as a "six pack", was also adopted by commercial aviation. After the Second World War the arrangement was changed to: (top row) airspeed, artificial horizon, altimeter, (bottom row) turn and bank indicator, heading indicator, vertical speed.

Further development

Of the old basic six instruments, the turn and bank indicator is now obsolete. The instrument was included, but it was of little use in the first generation of jet airliners. It was removed from many aircraft prior to glass cockpits becoming available. With an improved artificial horizon, including gyros and flight directors, the turn and bank indicator became needless except when performing certain types of aerobatics (which would not be intentionally performed in IMC to begin with). But the other five flight instruments, sometimes known as "the big five", are still included in all cockpits. The way of displaying them has changed over time, though. In glass cockpits the flight instruments are shown on monitors. But the display is not shown by numbers, but as images of analog instruments.

The artificial horizon is given a central place in the monitor, with a heading indicator just below (usually this is displayed only as a part of the compass). The indicated airspeed, altimeter, and vertical speed indicator are displayed as columns with the indicated airspeed and altitude to the right of the horizon and the vertical speed to the left in the same pattern as in most older style "clock cockpits".

Different significance and some other instrumentation

In good weather a pilot can fly by looking out the window. However, when flying in cloud at least one gyroscopic instrument is necessary to orientate the aircraft, being either an artificial horizon, turn and slip, or a gyro compass.

The vertical speed indicator, or VSI, is more of "a good help" than absolutely essential. On jet aircraft it displays the vertical speed in thousands of feet per minute, usually in the range -6 to +6. The gyrocompass can be used for navigation, but it is indeed a flight instrument as well. It is needed to control the adjustment of the heading, to be the same as the heading of the landing runway. Indicated airspeed, or IAS, is the second most important instrument and indicates the airspeed very accurately in the range of 45 to 250 knots. At higher altitude a MACH-meter is used instead, to prevent the aircraft from overspeed. An instrument called true airspeed, or TAS, exists on some aircraft. TAS shows airspeed in knots in the range from 200 knots and higher. (It is like the Mach-meter: not really a flight instrument). The altimeter displays the altitude in feet, but must be corrected to local air pressure at the landing airport. The altimeter may be adjusted to show an altitude of zero feet on the runway, but far more common is to adjust the altimeter to show the actual altitude when the aircraft has landed. In the latter case pilots must keep the runway elevation in mind. However a radio altimeter (displaying the height above the ground if lower than around 2000–2500 feet) has been standard for decades. This instrument is however not among the "big five", but must still be considered as a flight instrument.

(9,736 symbols)

http://www.readbook5.com/aircraft-digital-electronic-and-computer-systems/

TEXT 30 MOBILE SEARCH ENGINE OPTIMIZATION

Millions of users these days access the web using smartphones running on Android, iOS, or Windows. Hence, it has become imperative that websites adapt themselves to this changing environment and make suitable changes in their website design to attract more viewership.

The desktop version of a site might be difficult to view and use on a mobile device. The version that is not mobile-friendly requires the user to pinch or zoom in order to read the content. Users find this a frustrating experience and are likely to abandon the site. In contrast, a mobile-friendly version is readable and immediately usable. A recent Google update makes it mandatory that a website should be mobile-friendly to be effective on Mobile Search Engines. Note that a website that is not mobile-friendly will not have any impact on regular search engines either.

What is Mobile SEO?

Mobile Search Engine Optimization is the process of designing a website to make it suitable for viewing on mobile devices of different screen sizes having low bandwidth. Apart from following all the SEO rules which are applicable to a desktop website, we need to take additional care while designing a website for mobile devices. A website is mobile friendly if it has the following attributes:

- A good mobile website has a responsive design which performs well on desktops as well as mobile devices. It not only reduces the maintenance of the website but also makes the content consistent for the search engines.
- The contents of a good mobile website are easy to read on a mobile device without having to zoom the screen. It has appropriate fonts, colors, and layouts.
- It is easy to navigate through a good mobile website on a small screen. It provides links and buttons that can be easily maneuvered using a finger.
- A good mobile website is lightweight such that it takes less bandwidth and time to load on mobile networks.
- The Home Page of a mobile website plays the most important role in connecting users to the content they are looking for. Therefore, good mobile websites make sure the most important links are displayed on the Home Page so that they get enough visibility.

The ranking of a website depends heavily on how user friendly it is. You can follow the guidelines given below to design a great mobile-friendly website.

Optimize Your Site for Mobile

If your site is already optimized for search engines, then it should not be too difficult to optimize it for mobile devices. First, let us understand what it takes to go mobile. We can categorize the steps into three broad categories −

Step 1 − Select a Mobile Configuration

Step 2 − Inform Search Engines

Step 3 − Avoid Common Mistakes

Select a Mobile Configuration

There are three different mobile configurations that you can choose from −

Step 1 − Responsive Web Design

Step 2 − Dynamic Serving

Step 3 − Separate URLs

Each has its own advantages and disadvantages. Google recommends responsive design, however it supports all three configurations. The following table shows how the mobile configuration affects your URL and HTML code –

Mobile Configuration	URL	HTML
Responsive Web Design	Stays the same	Stays the same
Dynamic Serving	Stays the same	Different HTMLs
Separate URLs	Different HTMLs	Different HTMLs

Responsive Web Design

Google recommends responsive web design become it is the simplest mobile configuration and very easy to implement. It serves the same HTML code on the same URL, however it adjusts the display based on the screen size of the mobile device.

Dynamic Serving

Dynamic serving is a type of mobile configuration where the URL of your website remains unchanged, but it serves different HTML content when accessed from a mobile device.

When your content is dynamically served from the server, make sure you inform Google that the content it is crawling may look different on mobile devices. A major drawback of this approach is that you will have to do additional processing on your content at the server level before severing

it to the user. This approach puts unnecessary load on your server and makes it slow.

Separate URLs

When you maintain two different URLs – one for mobile users and another for desktop users – make sure you inform Google explicitly when to serve which version. Google does not recommend separate URLs because it can detect automatically that your mobile pages are different from your desktop pages.

This approach is not practical when you have a big website because maintaining two versions of the same website will require double the effort and money. At the same time, you cannot avoid various discrepancies in your content while maintaining two versions.

From the viewpoint of SEO, each URL performs separately. Hence your desktop ranking will never be added to the mobile ranking and they will always be assumed as separate websites. We don't recommend maintaining different URLs for mobile and desktop versions if you want to draw the benefits of SEO.

Inform Search Engines

Make sure Google and other search engines understand your mobile configuration. Most important of all, Google must understand your page so that it can rank your website properly. How you inform Google depends on which mobile configuration – responsive web design, dynamic serving, or separate URLs – you have opted for.

In case your site has a responsive design, Google's algorithms can understand it automatically without you having to inform Google. When you have a responsive design, just make sure you have the following meta-tag in your webpage header −

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

The viewport decides how your webpage will be displayed on a device. A site with responsive design varies its size based on the size of the device screen. Declare a viewport so that your webpage displays correctly on any device.

If your website is dynamically served, make sure you allow Google detect your configuration using the Vary HTTP header −

```
Vary: User-Agent
```

The Vary header is important to tell the search engines that different content will be served on desktops and mobile devices. This header is really important when your content is served by any cache system like a Content Delivery Network and those systems will make use of this header while serving content on different devices.

In case you maintain separate URLs, e.g., example.com and m.example.com, then you can inform Google by adding a special link rel=alternate tag in your desktop version and vice versa as follows.

Desktop page should have following in its header:

```
<link rel="alternate" media="only screen and (max-width: 640px)"
href="http://m.example.com" >
```

Mobile page should have following in its header:

```
<link rel="canonical" href="http://www.example.com" >
```

Avoid Common Mistakes

In order to optimize your website for mobile devices, make sure you avoid committing the following mistakes −

- Slow Mobile Pages − Mobile networks are slower as compared to wired Internet networks, so it is important to pay attention to how fast your mobile pages load. It is a critical Google ranking factor. Use a mobile SEO tool to find out your mobile page speed. Google provides a number of good tools that you can use. Browse the following link − *https://www.google.com/webmasters/tools/mobile-friendly/*
- Don't Block CSS and JavaScript − Google recommends to use inline CSS and Javascripts for mobile friendly websites so that they can be downloaded along with the content. So if you don’t have much CSS, then try to adjust it within the tag itself; but if you are using a lot of CSS in separate files, then try to include it at the bottom which will stop blocking the other content being downloaded. The same rule applies to Javascript, which can be kept inside the page itself or included at the bottom of the page. If you can avoid including the file at the top of the page, then make use of async attribute while including them.

```
<script async type="text/javascript" src="jquery.js"></script>
```

- Mobile Redirects − Since mobile networks are normally slow, too many redirects can hurt your page speed. If you are maintaining multiple URLs, make sure all your links point to the relevant pages. In case you maintain multiple URLs and you recognize a user is visiting a desktop page from a mobile device and you have an equivalent mobile page at a different URL, then redirect the user to that URL instead of displaying a 404 error.
- Heavy Images − Heavy images reduce the load time, however we cannot completely get rid of them since they are useful and effective. Therefore you should maintain a good balance between text and heavy images. Use a good tool to optimize your images and save them at low resolution to avoid heavy downloads.

•Avoid plug-ins and pop-ups − Plug-ins like Flash and Java may not be available on user's mobile device. Always ensure you don't have any unplayable content on your mobile page. Avoid using pop-ups on mobile pages because it becomes quite clumsy to close these pop-ups on a mobile device.

While creating a mobile page, always keep in mind that the user has limited space to work on. So, you need to be as concise as possible while creating titles, URLs, and meta-descriptions – of course without compromising the essence or quality of information.

Useful Tools

Here is a list of some useful tools that you can use to find out how mobile friendly your site is:

Google Webmaster Tools − Use the available Google tools and techniques to understand what should be used and what should be avoided while designing desktop as well as mobile websites.

Mobile Emulator − It lets you see how your site appears on a wide variety of mobile devices.

Moz Local − Use this tool to ensure that your local SEO is in order.

Responsive Web Design Testing Tool − Use this tool to see how your responsive site looks like on a variety of mobile devices with different standard screen sizes.

Screaming Frog − This is a useful tool that allows you to analyze your site and double-check all the redirects.

User Agent Switcher − This is a Firefox add-on that you can use to find out how your site looks like when accessed from a different user agent.

(8,172 symbols)

https://www.tutorialspoint.com/seo/mobile-seo-techniques.htm

TEXT 31 INTERNET OF THINGS: THE TOOLS, PLATFORMS AND PROGRAMS YOU NEED TO KNOW

IoT development projects are everywhere, and affordable, advanced technology is the driving force behind this fast-growing phenomenon. Smaller, more accessible hardware and the flexibility to use common programming languages make it easier than ever before to develop these embedded IoT systems. From hobbyists programming their own single-board computers to companies developing devices we can control from our mobile devices, the IoT is rapidly expanding.

Whether you're creating a quick prototype or an entire IoT-powered business application, here's a look at the small but incredibly smart technology of IoT development. We'll cover IoT data, hardware, and software considerations, plus the most popular IoT skills on the rise so you'll know what to look for when seeking out top talent for your IoT initiative.

Top IoT skills on the rise

The IoT has attracted the attention of companies all across the globe, with many creating internal business units dedicated to IoT development. According to a recent survey from analyst firm, Gartner, 43 percent of organizations are using or plan to implement IoT in 2016.

IoT has become a top business initiative for many companies. So who is the talent who will be driving this new technology, and what skills will they need to have? Here's a look at the top growing global IoT categories and skills on the Upwork platform to give you an idea of the scope and demands of IoT projects.

Data science and analytics – 1027%*

- Data mining: 230%
- Machine Learning: 199%
- Matlab: 78%

IT & Networking – 120%

- Computer networking: 91%
- Network security: 46%
- Linux system administration: 26%

Engineering & Architecture – 68%

- Circuit design: 231%
- AutoCAD: 217%
- 3D design: 29%

Wearables – 68%

- Electrical engineering: 159%
- GPS development: 66%
- 3D design: 29%

Security – 51%

- Security infrastructure: 194%
- Security engineering: 124%
- Network security: 46%

Connected Home – 41%

- Raspberry Pi: 17100%*
- Circuit design: 231%

- Microcontroller programming: 225%

Web, Mobile and Software Development – 40%

- Node.js: 86%
- MongoDB: 63%
- iPhone app development: 40%

Big data, data storage and processing/backend programming – 17%

- Apache Spark: 1667%*
- Big data: 183%
- MongoDB: 63%

Note: Data is sourced from the Upwork database and is based on the number of job posts on Upwork from October 2014–December 2015.

**Percentages reflect newer skills which have grown more quickly on the site*

Developing an IoT device or distributed IoT service

From a development standpoint, creating IoT devices hinges on embedded programming. There are both software and hardware angles to consider when creating an IoT prototype – the small computer embedded in the object or device, and the software that makes it run. As mentioned above, this includes things like wearables, connected home devices, circuit design, GPS programming, 3D design, and more.

Fortunately, many of these software systems and software development kits (SDKs) now use programming languages and operating systems that engineers already use for mobile and web development, which opens the field up to many more developers.

If you're creating a fully fledged distributed IoT service, there are many angles to consider: development of the embedded device itself, the IT and networking services that power it, data and analytics, and design and development of an integrated UI (e.g., a mobile app to control your home's thermostat).

You'll need to:

- Choose your hardware platform (i.e., your processing board)
- Develop the application software, including any back-end and networking support
- Create the integrated UI
- Develop the APIs, beacons, web sockets, and procedure calls that enable the high-level communications that occur between devices
- Establish security, data storage, and analytics measures

IoT development platforms

To get started, you'll need a platform for the product development team to develop and launch the product on.

One incredibly popular hardware/software platform for creating interactive IoT objects and devices is the Arduino platform, which includes a physical board processor, shields with individual libraries of C code, and an integrated development environment (IDE) for writing, compiling, and uploading code.

Windows has also gotten into the IoT game with Windows 10 IoT Core, an IoT-optimized version of Windows 10 that uses Visual Studio and the Arduino Wiring API. It runs on a few different boards, including Raspberry Pi 2. IBM has launched Quarks IoT tools, another enterprise-grade option.

IOT hardware & operating systems

The range of embedded devices is vast – from small prototypes people develop for fun, to mass-produced technology – and there's hardware to suit every project. Usually, these small computers are referred to as boards, or chips, and they come with a wide range of price points and processing capabilities.

Hardware components can include low-power boards; single-board processors like the Arduino Uno**;** field-programmable gate arrays (FPGA); and shields, which are smaller boards that plug into main boards to extend functionality by abstracting specific functions (e.g., GPS, light and heat sensors, or interactive displays). A programmer will specify a board's inputs and outputs, then create a circuit design schematic to determine how these inputs and outputs interact.

Another well-known IoT platform is Raspberry Pi 2, a "tiny affordable computer" that can house a web server that fits in the palm of your hand. Often shortened to just "RasPi," it has enough processing power and memory to run Windows 10 IoT Core. RasPi is great for more heavy-duty processing, especially when using the Python programming language.

BeagleBoard is a single-board computer with a Linux-based OS that uses an ARM processor. They're capable of even more powerful processing than RasPi, and have a price tag to match. Tech giant Intel's Galileo and Edison boards are other options, both great for larger scale production, and Qualcomm has manufactured an array of enterprise-level IoT technology for cars and cameras to healthcare. Samsung's ARTIK platform has three circuit boards, with small ones for wearables and a larger 8-processor chip capable of video functionality.

This is just a glimpse at some of the technology that's out there; an IoT pro can help recommend the kind of power and operating system you'll need on the hardware side that's appropriate for your device or prototype.

Embedded eyes and ears: sensor and beacon technology

Bluetooth beacons embedded within devices allow IoT objects to broadcast information to nearby mobile devices. These low-power sensors with technology like Bluetooth Low Energy (BLE) – the one-way communication from objects to nearby devices – let our mobile phones listen for signals when we're close to an IoT object. BLE is different from traditional Bluetooth technology in that it's cheaper, requires less power (one beacon can go three years without a charge), and is ideal for simple applications and quick pops of data, like sending a coupon to a nearby mobile phone.

In 2013, Apple launched iBeacon, a low-power bluetooth sensor that can be embedded in objects and picked up by nearby iOS or Android devices running apps that have been programmed with the Core Location APIs. Another popular BLE beacon is AltBeacon, a free option with a bit more data capacity than iBeacon. While both iBeacon and AltBeacon rely on databases for their functionality, Google's URIBeacon project delivers URLs (similar to a QR code) rather than packets of information from a database, so it's easier to update, reconfigure, and has the entire web as its database.

IOT software & programming languages

IoT programming languages used to be unique to embedded systems, but now this software uses more common languages that web developers already know and use. So how do you choose which language to use for your IoT project?

First, embedded systems have a certain set of limitations to consider – low processing power, and smaller amounts of RAM and storage. The most commonly used operating systems for these embedded computers are Linux or UNIX-like OSs like Ubuntu Core or Android. While you may have to decide based on your chosen hardware platform, you also can opt for a language your developer is already familiar with, or decide based on factors like its compatibility with your IoT ecosystem, the size and memory of the code, efficiency requirements, or speed of development.

IoT programming languages range from general-purpose languages like C++ and Java to embedded-specific choices like Google's Go language or Parasail. Each offers a few advantages and disadvantages.

Your developer will be able to advise you which is best, but here's a quick overview.

1. **C & C++:** The C programming language has its roots in embedded systems—it even got its start for programming telephone switches. It's pretty ubiquitous, and many programmers know it. C++ is the object-oriented version of C, popular for both the Linux OS and Arduino embedded IoT software systems. Both languages have an advantage because they were designed to be written specifically for the hardware they're running on, so you can accomplish the fine-tuned coding ideal for embedded systems.
2. **Java:** Where Java has an advantage over C and C++ is that the code is less hardware-specific, making it more portable. It requires libraries to run on different hardware, but once you've invested in that code base, you're all set – it's the "write once, run anywhere" language.
3. **Node.js and JavaScript:** JavaScript is a great option for IoT. Node.js code can run a complete IoT system, running on both an embedded smart device and the server-side software that's powering it. It's an interpreted language, however, making it a better match for more robust embedded systems, like Raspberry Pi. DeviceJS is a JavaScript-based development platform for programming sensors and controlling devices.
4. **Python:** Python has become one of the "go-to" languages in Web development, and its use has spread to the embedded control and IoT world – specifically the Raspberry Pi processor. Python is an interpreted language, which makes it flexible, easy to read, and quick to write. Plus, it's a powerhouse for data-heavy applications.
5. **Languages designed for I/O programming** include Go from Google, Rust from Mozilla, Forth, and Parasail – a language designed specifically for embedded programming.
6. **B#:** Unlike most of the languages mentioned so far, B# hasn't been retrofitted for embedded systems, it was designed for them. It's small and fast, and can run on smaller hardware platforms thanks to its 24k memory size.

IOT data and security considerations

While the IoT opens up amazing new possibilities, it also opens up new security concerns. Anytime we're advancing the way we monitor, detect, and track ourselves and the things around us, what we do with the data – and how it's sent across networks – can get sensitive. That's why security needs to be incorporated at every stage to keep hackers at bay.

An article in Sophos pointed out security vulnerabilities with Wi-Fi-connected Ring doorbells, noting, "If you're a programmer, and

you're enabling your latest electronic gadget to join the IoT, remember to think security, even if you never expect that device to be installed on the public-facing internet."

This means it's important to take certain programming steps (and avoid certain security shortcuts) like proxies and encryption, to keep hackers from using devices to access a user's personal network.

(9,225 symbols)

https://www.upwork.com/hiring/development/internet-of-things-platforms-and-programs-you-need-to-know/

TEXT 32 PYTHON: A POWERFUL LANGUAGE FOR HIGH-TRAFFIC, DATA-HEAVY APPS

Python is a widely used, general-purpose, high-level back-end programming language that's highly valued by startups who need to quickly prototype and develop applications, as well as data-driven companies that need to integrate data analysis and statistical techniques into their workflows.

Its combination of readability, flexibility, and suitability to data science operations have made Python one of the most popular and beloved languages according to developers on Stack Overflow. In this article, we'll explore what sets Python apart from other programming languages, why it's popular with data scientists, and what you should look for in a Python engineer.

High-level, readable, and efficient

One of Python's defining characteristics is its efficiency. Every programming language has to balance the programmer's time and the machine's resources. Python is biased toward the former, with a guiding philosophy that comes down to **"**there should be one – and preferably only one – obvious way to do something." That can mean there's a bit of a learning curve as developers learn the ins and outs of Python syntax, but the upside is that developers can do more with fewer lines of code compared to more lower-level implementation languages like Java or C++. This efficiency is especially valuable for startups who need to quickly prototype applications and get them to market.

Python is also famous for its code readability, meaning that an application written by a developer in Python is more likely to be intelligible to the developers who have to maintain it months and years down the line.

Beloved by data scientists

Along with R and Java, Python is one of the most popular languages for data science and statistical analysis. For data scientists, Python combines Java's suitability for building high-traffic web applications with R's focus on executing complex statistical functions.

Another one of Python's strongest assets is its extensive set of libraries. These libraries can make it easier for developers to perform complex machine learning or statistical analysis tasks without having to rewrite many lines of code. Some of the most popular libraries include tools for data manipulation and visualization (NumPy, SciPy, and matplotlib), data mining and Natural Language Processing (Pattern, NLTK). Perhaps unsurprisingly, Python is the language of choice for organizations with data-heavy workflows, from YouTube to the New York Stock Exchange to the National Web Service.

Python basics

- It's object-oriented**.**
- It's cross-platform, working on Linux, Windows, Mac, and most other operating systems.
- Python's standard library supports:
  - HTML & XML
  - JSON
  - E-mail processing
  - HTTP Server libraries, easy for developing servers, and support for FTP, IMAP, and other Internet protocols
- It's free and supported by an active open-source community.
- It's often substituted for PHP in the LAMP software stack.

Popular Python frameworks

Python engineers have a number of options when it comes to frameworks. Frameworks are collections of packages that take care of the implementation details so you can quickly write applications. Which framework is best for your project depends on the scale of your application, its complexity, and your data needs.

- **Django:** A very structured, all-in-one framework with lots of "scaffolding," it's designed for large-scale, complex applications. Lots of components and elegant database management make it a good choice for data-heavy sites.
- **Flask:** A lightweight, minimalist framework, it gives developers a more flexible approach to using Python. Similar to Pyramid, it has a loose development style and is ideal for smaller, less complicated applications.

- **Pyramid:** The middle road of Flask and Django, this framework offers a mix of flexibility and structure and is also good for complicated, bigger applications.
- **Twisted:** A low-level networking Python framework.
- **Tornado:** A framework that's good for web servers and web apps.

The Python developer's toolbox

What should you look for in a Python engineer? Experience on large-scale, high-traffic applications is at the top of the list, along with fluency in SQL and database optimization. Common duties and core skills of Python developers include modular programming, object-oriented programming, and extensive experience with SQL.

Other related skills and technologies a Python engineer should know include:

- Unix/Linux operating systems
- Frameworks: Django, Flask, or Pyramid
- MVC pattern
- HTML and XML
- Strong SQL knowledge and relational database design understanding, with familiarity of MySQL, MS SQL, or Postgres
- Experience with web-based user interfaces, including RESTful APIs
- Back-end cloud applications and web services
- Object-oriented programming
- the LAMP software stack.

(3,953 symbols)

https://www.upwork.com/hiring/development/python-programming-language/

TEXT 33 INSIDE IT SECURITY: HOW TO PROTECT YOUR NETWORK FROM EVERY ANGLE

Network security. Cyber security. Endpoint security. These different, often overlapping arms of IT security can get confusing. As hackers get smarter, it's increasingly important to know what each does and how to implement them into your own network.

In the wake of the highly-connected Internet of Things (IoT) and the rise of the cloud, we're facing increased vulnerabilities to our networks – networks that are less monolithic, legacy architectures and more distributed**,** microservice-based networks. With large-scale data breaches

making headlines, whether you're a small startup or an enterprise organization, security should be a top priority.

In this article, we'll explore the different types of IT security and what technologies and methods are used to secure each so you can arm your network with the people and plans you need to have excellent lines of defense in place and keep attacks at bay.

The IT security chain

Why are there so many types of IT security? The more links in a network's chain, the more opportunities for hackers to find their way in. Each component requires its own subsequent security measures – with many of them overlapping and working in tandem, much like the actual components of a network do.

It's also important to note that with security, there's no one-size-fits-all approach. Every network is different and requires skilled professionals to create tailored plans across all fronts: apps, databases, network devices, cloud servers, IT infrastructures, and the often weakest link in the security chain: users. These security plans are living, breathing things that need to be updated, upgraded, and patched on a constant basis, too.

Let's start broad and work our way into narrower fields of security.

Information security and information technology (IT) security sound similar, and are often used interchangeably, but they're slightly different fields. When we're talking about information security (or infosec), we're actually referring to protecting our data – whether that's physical or digital. IT security is a bit more specific in that it's only referring to *digital* information security.

IT security pretty much covers all of the types of security within a network, from components like databases and cloud servers to applications and the users remotely accessing the network. They all fall under the IT security umbrella.

Within this is another term to know: information assurance. This means that any important data won't be lost or stolen in the event of an attack or a disaster – whether that's a tornado wiping out a server center or hackers breaking into a database. It's commonly addressed with things like backups and offsite backup databases and rests on three main pillars: confidentiality, integrity, and availability (CIA). These philosophies carry over into every other aspect of security, whether it's application security or wireless security.

IT security experts (also, system administrators and network admins, which we'll talk about next) are one of the most important team members

you can hire. They're responsible for the safety and security of all of a company's hardware, software, and assets, and regularly audit back-end systems to ensure they're airtight. Through security analysis, they can identify potential security problems and create "protect, detect, and react" security plans.

Network security: the best defenses

Network security is anything you do to protect your network, both hardware and software. Network administrators (or system administrators) are responsible for making sure the usability, reliability, and integrity of your network remains intact. A hacker is capable of getting into a network and blocking your access, for example by holding a system hostage for a bitcoin ransom. You need an excellent defense in place to ensure you're protected.

Detecting weaknesses in a network can be achieved through:

- **Security engineering:** the practice of protecting against these threats by building networks to be safe, dependable, and secure against malicious attacks. Security engineers design systems from the ground up, protecting the right things in the right ways. If a software engineer's goal is to ensure things do happen (click here, and this happens), a security engineer's goal is to ensure things *don't* happen by designing, implementing, and testing complete and secure systems.

As a part of security engineering, there are proactive measures to predict where vulnerabilities might lie and reinforce them before they're hacked:

- **Vulnerability assessment:** Engineers identify the worst case scenarios and set up proactive plans. With security analysis software, vulnerabilities in a computer, network, or communications infrastructure are identified and addressed.
- **Penetration testing**: This entails deliberately probing a network or system for weaknesses.
- **Network intrusion detection systems (NIDS):** This type of software monitors a system for suspicious or malicious activity.

Network admins are able to target threats (whether through suspicious activity or large queries to a database), then halt those attacks, whether they're passive (port scanning) or active, like:

- **Zero-day attacks**, also called zero-hour attacks – attacks on software vulnerabilities that often occur before the software vendor is aware of it and can offer a patch. Or, hackers will initiate attacks on the

software vulnerability the day that it's made public there's an issue, before users can install patches (hence the name "zero day")

- Denial of service attacks
- Data interception and theft
- Identity theft
- SQL injection

Other methods of protecting networks include:

- **IT Security frameworks:** These act like blueprints for a company to set up processes and policies for managing security in an enterprise setting. Which a company uses can depend on the industry and compliance requirements. COBIT is popular among larger, publicly traded companies, ISO 27000 Series is a broad set of standards that can be applied to a number of industries, and NIST's SP 800 Series is used in government industries, but can be applied elsewhere.
- **Password "salt and peppering"**: Adding a salt, or random data, to a password makes common passwords less common. A pepper is also a random value attached to the password, which is helpful in slowing hackers down.
- **Authorization, authentication,** and **two-factor authentication** (sometimes sent via SMS, although this can prove vulnerable as well)
- **Virtual Private Networks (VPNs)**
- **Application whitelisting**, which prevents unauthorized apps from running on a computer
- **Firewalls:** Block unauthorized access to a network or data interceptions
- **Honeypots:** These are like decoy databases that attract hackers but don't house any important information.
- **Anti-virus software**
- **Encryption –** decoding data, in transit or at rest, including end-to-end encryption often used in messaging apps and platforms that only allows encrypted messages to be read by sender and receiver

Within network security is also content security, which involves strategies to protect sensitive information on the network to avoid legal or confidentiality concerns, or to keep it from being stolen or reproduced illegally. Content security largely depends on what information your business deals in.

Endpoint security: securing the weakest link

It's said that users are often the weakest link in the security chain, whether it's because they're not properly educated about phishing

campaigns, mistakenly give credentials to unauthorized users, download malware (malicious software), or use weak passwords. That's why endpoint security is so crucial – it protects you from the outside in.

Endpoint security technology is all about securing the data at the place where it both enters and leaves the network. It's a device-level approach to network protection that requires any device remotely accessing a corporate network to be authorized, or it will be blocked from accessing the network. Whether it's a smartphone, PC, a wireless point-of-sale, or a laptop, every device accessing the network is a potential entry point for an outside threat. Endpoint security sets policies to prevent attacks, and endpoint security software enforces these policies.

If you've ever accessed a network through a virtual private network (VPN), you've seen endpoint security in action. Malware is one of the core threats addressed by endpoint security, including remote access trojans (RATs), which can hack into a laptop and allow hackers to watch you through your webcam.

Internet security: guarding against cyber crimes

The internet itself is considered an unsecured network – a scary truth when we realize it's essentially the backbone for how we give and receive information. That's where internet security (or cyber security) comes in, and it's a term that can get pretty broad, as well. This branch of security is technically a part of computer security that deals specifically with the way information is sent and received in browsers. It's also related to network security and how networks interact with web-based applications.

To protect us against unwittingly sharing our private information all over the web, there are different standards and protocols for how information is sent over the internet. There are ways to block intrusions with firewalls, anti-malware, and anti-spyware – anything designed to monitor incoming internet traffic for unwanted traffic or malware like spyware, adware, or Trojans. If these measures don't stop hackers from getting through, encryption can make it harder for them to do much with your data by encoding it in a way that only authorized users can decrypt, whether that data is in transit between computers, browsers, and websites, or at rest on servers and databases.

To create secure communication channels, internet security pros can implement TCP/IP protocols (with cryptography measures woven in), and encryption protocols like a Secure Sockets Layer (SSL), or a Transport Layer Security (TLS).

Other things to have in an internet security arsenal include:

- Forms of email security
- SSL certificates
- WebSockets
- HTTPS (encrypted transfer protocols)
- OAuth 2.0, a leading authorization security technology
- Security tokens
- Security software suites, anti-malware, and password managers
- Frequently updating and installing security updates to software, e.g., Adobe Flash Player updates
- Encryption, and end-to-end encryption

Cloud security: protecting data that's here, there, and everywhere

Much of what we do over the web now is cloud-based. We have cloud-based servers, email, data storage, applications, and computing, which means all of the communication between onsite and the cloud needs to be secure, too. With all of this connectivity and the flowing of (sometimes sensitive) information comes new concerns with privacy and reliability – and the cloud can be *notoriously* vulnerable. This has given way to a new sub-domain of security policies: cloud computing security**.**

Computer security, network security, and information security as a whole all need to be optimized for the cloud. For businesses that use public clouds, private clouds, or a hybrid cloud – information is getting exchanged between the two regularly and needs to be protected.

Building a cloud security framework involves creating a strategic framework for how all operations will happen in a cloud environment, managing access, protecting data, and more.

Application security: coding apps to be safe from the ground up

A lot of the internet security focus is on patching vulnerabilities in web browsers and operating systems, but don't neglect application security – a majority of internet-based vulnerabilities come from applications. By coding applications to be more secure from the start, you're adding a more granular layer of protection to your internet and network security efforts, and saving yourself a lot of time and money.

App security does rest on top of many of the types of security mentioned above, but it also stands on its own because it's specifically concerned with eliminating gaps and vulnerabilities in software at the design, development, and deployment stages. Security testing (which should be conducted throughout the code's lifecycle) digs through the app's code for vulnerabilities, and can be automated during your software development cycle.

Choosing a language, framework, and platform with extra security fortifications built in is paramount, too. For example, Microsoft's .NET framework has a lot of built-in security, and the Python Django-style Playdoh platform addresses application security risks. Rising in popularity is the Spring Security framework, a Java framework known for excellent built-in authentication and authorization measures, and the PHP framework Yii prioritizes security, as well.

Aside from framework choice, there are a few strategies to bolster application security, including:

- Ensuring TLS
- Authentication and authorization measures
- Data encryption
- Sandboxing applications
- Secure API access
- Session handling

By adopting a proactive security stance, educating your users, and taking advantage of the latest in authentication measures, you'll be better able to prevent, detect, and strengthen your company against attacks. However, it's important to remember that securing your network isn't a one-time thing—it's an ongoing process that needs to be constantly occurring and evolving along with your website and organization to ensure you're protected in the face of the ever-changing landscape of security threats.

(11,119 symbols)

https://www.upwork.com/hiring/development/understanding-it-security-and-network-security/

TEXT 34 ENCRYPTION BASICS: HOW IT WORKS & WHY YOU NEED IT

We've entered a time when the conveniences of widespread connectivity, including the cloud, have put us at more risk than ever of getting hacked. When data does fall into the wrong hands, the consequences can be devastating. High-profile data breaches and ransomware attacks have organizations and individuals on red alert for the best ways to safeguard their data and networks, both now and for the future.

While good IT security strategies can be very effective in protecting networks – essentially letting the good guys in and keeping the bad guys

out – how do you account for all of the data that's traveling across the airwaves between mobile devices, browsers, databases, and the cloud?

There's a time-tested science that is increasingly becoming a crucial link in the security chain: encryption. Encryption scrambles text to make it unreadable by anyone other than those with the keys to decode it, and it's becoming less of an added option and more of a must-have element in any security strategy for its ability to slow down and even deter hackers from stealing sensitive information. If good encryption is capable of hindering investigations by FBI experts, consider what it could do for you and your company's sensitive information.

If you've been putting off adopting encryption as a part of your security policy, delay no more. Here's a guide to the science of encryption, and how you can begin implementing an encryption strategy today.

What is encryption and how does it work?

While IT security seeks to protect our physical assets – networked computers, databases, servers, etc. – encryption protects the data that lives on and between those assets. It's one of the most powerful ways to keep your data safe, and while it isn't impenetrable, it's a major deterrent to hackers. Even if data does end up getting stolen, it will be unreadable and nearly useless if it's encrypted.

How does it work? Encryption – based on the ancient art of cryptography – uses computers and algorithms to turn plain text into an unreadable, jumbled code. To decrypt that ciphertext into plaintext, you need an encryption key, a series of bits that decode the text. The key is something only you or the intended recipient has in their possession. Computers are capable of breaking encrypted code by guessing an encryption key, but for very sophisticated algorithms like an elliptic curve algorithm, this could take a very, very long time.

Here's a very simple example. Say you want to encrypt this sentence:

"Protect your data with encryption".

If you use a 39-bit encryption key, the encrypted sentence would look like this:

"EnCt210a37f599cb5b5c0db6cd47a6da0dc9b728e2f8c10a37f599cb5b5c0db6cd47asQ
K8W/ikwIb97tVolfr9/Jbq5NU42GJGFEU/N5j9UEuWPCZUyVAsZQisvMxl9h9IwEmS."

You can send that encrypted message to someone, separately share the key, then they're able to decrypt it and read the original sentence.

If you send an encrypted email, only the person with the encryption key can read it. If you're using an encrypted internet connection to shop online, your information and credit card number are hidden from unauthorized users, like hackers, illegal surveillance, or identity thieves.

If you encrypt data before syncing it with the cloud, the cloud – or anyone breaking into it – can't read that data. Even iPhones are encrypted to protect their data if they're lost or stolen – something that has made headlines when organizations like the FBI or the NSA need access to them for investigations.

But encryption can be used for bad, too. Ransomware attacks are becoming more prevalent, also called denial of service (DOS) attacks that use encryption software to lock users out of their computers until they pay a fee.

Encrypting data "In transit" vs. Data "At rest"

Basically, the data we encrypt is always either:

- **In transit**, meaning it's moving via email, in apps, or through browsers and other web connections
- **At rest**, when data is stored in databases, the cloud, computer hard drives, or mobile devices

Encrypting this data is achieved mainly through:

1. **Full disk encryption (FDE)**: the primary way to protect computer hard drives and the at-rest data on them. Any files saved to the disk (or an external hard drive) are automatically encrypted. There are intermediate options for disk encryption, as well – folder encryption, volume encryption, etc. – that aren't quite full-disk encryption, but in between.
2. **File encryption**: a way to encrypt at-rest data on a file-by-file basis so it cannot be read if intercepted. This isn't automatic, but it's beneficial because that data will stay encrypted after it's left its place of origin.
3. **End-to-end (E2E) encryption**: obscures any content of messages so only senders and receivers can read it, like the early **Pretty Good Privacy (PGP)** email encryption software. The idea with E2E encryption is that it tackles all the vulnerabilities on the communication chain: the middle (intercepting a message during delivery), and both ends (sender and receiver). This is not just a niche offering anymore, either – platforms like Facebook Messenger and Apple's iMessage have E2E encryption now, too.
4. **Encrypted web connections**: via HTTPS, encrypted web connections use a Secure Sockets Layer (SSL) or transport layer security (TLS) protocols. With secure internet connections, we're able to have better protected communications on the web. These aren't impenetrable, but there's less risk of exploit. *How it works:* HTTPS uses SSL and TLS

certificates when a browser and server communicate over the web. These are encryption keys, and when both browser and server have them, they're authorized to access the encrypted data that's passed between them. It's a very basic, but very important, security measure when connecting to the web. If you've ever seen "https" instead of "http," or noticed a lock in the URL bar of your browser, you're accessing a secure site.

5. **Encrypted email servers**: S/MIME (Secure/Multipurpose Internet Mail Extensions) public key encryption essentially gives SMTP (simple mail transfer protocol) email servers a leg up by allowing them to send and receive encrypted messages, not just simple text messages.

6. **Pre-encrypting data that's synced with the cloud**: there's plenty of software available that can pre-encrypt data before it even gets to the cloud, making it unreadable by the cloud or anyone who hacks into it. Note that any files still stored on the local machine aren't encrypted and are still vulnerable. This accounts only for files sent to the cloud encrypting tech.

Encryption can be simple, like secret-key, or incredibly complex, like the Advanced Encryption Standard (AES), depending on the algorithm and the length of the key. The longer the key, the more protection, but also the more processing power required to handle the encrypting and decrypting process.

A few types of encryption to know include:

- **Secret-key algorithms**: Also known as symmetric algorithms, or private-key, this algorithm uses the same key for encryption and decryption. This is a touch more vulnerable because anyone who gets a hold of that one key can read anything you encrypt. Also, passing that secret key over internet or network connections makes it more vulnerable to theft.
- **Public-key algorithms**: These are also known as asymmetric algorithms. With public-key encryption, there are two different, related encryption keys – one for encryption, and one for decryption. The public key is how the information is sent to you, and the private key decodes it (much like having a secure lock box on your front porch that a delivery person can put a package in, then only you can access that package with your private key). The benefit here is the key isn't subject to being sent over insecure networks, but it does require more computer processing power so it's a bit slower.
- **Block ciphers**: Like the Triple Data Encryption Standard (DES), or 3DES, these encrypt data a block at a time. Triple DES uses three keys and

is a pretty great encryption option for financial institutions that need to protect sensitive information.

- **Stream ciphers**: A symmetric algorithm, it uses a keystream, a series of randomized numbers, to encrypt plaintext one character at a time. Rabbit, W7, and RC4 are popular stream ciphers.
- **Elliptic curve cryptography**: A form of public-key encryption, it can be practically unbreakable for normal computers, or "hard". This is security industry speak for technology that's not completely unbreakable, but is generally accepted to be up to best standards.
- **Blockchain cryptography**: Blockchain technology is essentially a type of distributed database, best known as the basis for Bitcoin, that uses cryptography to safely store data about financial transactions. Blockchain cryptography is a form of "cryptocurrency", using public-key encryption, and it's valuable in its ability to provide direct, trustworthy and fraud-proof transactions between users on a peer-to-peer network. Because blockchain databases are distributed, they're more resilient in the face of a DOS attack, so more companies are exploring this.

A few popular algorithms include:

- **Advanced Encryption Standard (AES)**: A block cipher, this is pretty much the gold standard, per the U.S. Government. It offers 128-, 192-, and 256-bit encryption, the last two reserved for instances that require extra-strength protection.
- **RSA**: This asymmetric algorithm uses paired keys and is pretty standard for encrypting information sent over the internet, although it's been through some issues of getting broken, which have then been resolved.
- **IDEA (International Data Encryption Algorithm)**: This block cipher with a 128-bit key has a great track record for not being broken.
- **Signal Protocol**: This open-source encryption protocol is used for asynchronous messaging, like email.
- **Blowfish and Twofish**: Both of these block ciphers are free to use and popular among e-commerce platforms for protecting payment information. They were created by the same person and offer symmetric encryption with keys varying in bit length. Twofish is the successor and offers longer encryption keys.
- **Ring Learning With Errors or Ring-LWE**: This protocol ramps up elliptic curves by adding in a new type of encryption that might be unbreakable by quantum computers.

What is key management and why is it important?

Key management is another important aspect of encryption. Keys are how all of that encrypted data becomes readable, so how you handle them is just as sensitive as the data itself.

Many businesses worry about this aspect of encryption – after all, if you lose an encryption key, you lose access to your data, too. That's why key management dictates how keys are stored (and shared) so prying eyes can't get a hold of them, making your entire encryption schema moot.

- **Diffie-Hellman key exchange**: This secure way for people to create a key allows them to share secure information. This method is also touted as "**perfect forward secrecy**", meaning that theoretically, at no point in the future can messages encrypted with a Diffie-Hellman key be decrypted.
- **Double Ratchet algorithm**: Based on the above, the Double Ratchet algorithm is a key management algorithm used in end-to-end encryption of instant messaging, like the Signal messaging app.

This article just scratches the surface of the art and science of encryption, but hopefully it gives you enough basic understanding of this important security technology.

(9,204 symbols)

https://www.upwork.com/hiring/development/introduction-to-encryption-data-security/

TEXT 35 HOW CLOUD COMPUTING IS CHANGING THE SOFTWARE STACK

Are sites, applications, and IT infrastructures leaving the LAMP stack (Linux, Apache, MySQL, PHP) behind? How have the cloud and service-oriented, modular architectures facilitated the shift to a modern software stack?

As more engineers and startups are asking the question "Is the LAMP stack dead?" – on which the jury is still out – let's take a look at "site modernization," the rise of cloud-based services, and the other ever-changing building blocks of back-end technology.

From the LAMP era to the Cloud

Stackshare.io recently published its findings about the most popular components in many tech companies' software stacks these days – stacks that are better described as "ecosystems" due to their integrated,

interconnected array of modular components, software-as-a-service (SaaS) providers, and open-source tools, many of which are cloud-based.

It's an interesting shift. Traditional software stacks used to be pretty cut and dry. Acronyms like LAMP, WAMP, and MEAN neatly described a mix of onsite databases, servers, and operating systems built with server-side scripts and frameworks. When these systems grow too complex, though, the productivity they enable can be quickly eclipsed by the effort it takes to maintain them. This is up for debate, though, and anything that's built well from the ground up should be sturdy and scalable. However, a more modular stack approach still prompted many to make the shift.

A shift in the software stack status quo?

For the last five or so years, the monolith, LAMP-style approach has come more into question whether it's the best possible route. Companies are migrating data and servers to the cloud, opting for streamlined API-driven data exchange, and using SaaS and PaaS solutions as super-scalable ways to build applications. In addition, they're turning to a diverse array of technologies that can be more easily customized and integrated with one another – mainly JavaScript libraries and frameworks – allowing companies to be more nimble, and less reliant on big stack architectures.

But modularity is not without its complexities, and it's also not for everyone. SaaS, mobile, and cloud-computing companies are more likely to take a distributed approach, while financial, healthcare, big data, and e-commerce organizations are less likely to. With the right team, skills, and expectations, however, it can be a great fit.

New, scalable building blocks like Nginx, New Relic, Amazon EC2, and Redis are stealing the scene as tech teams work toward more modular, software-based ecosystems – and here are a few reasons why.

What are some of the key drivers of this shift?

1. Continuous deployment

What's the benefit of continuous deployment? Shorter concept-to-market development cycles that allow businesses to give customers new features faster, or adjust to what's happening with traffic.

It's possible to continuously deploy with a monolith architecture, but certain organizations are finding this easier to do beyond a LAMP-style architecture. Having autonomous microservices allows companies to deploy in chunks continuously, without dependencies and the risk of one failure causing another related failure. Tools like GitHub, Amazon EC2, and Heroku allow teams to continuously deploy software, for example, in an Agile sprint-style workflow.

2. The CLOUD is creating a new foundation

Cloud providers have completely shaken up the LAMP paradigm. Providers like Amazon Web Services (AWS) are creating entirely new foundations with cloud-based modules that don't require constant attention, upgrades, and fixes. Whereas stacks used to comprise a language (Perl, Python, or PHP), a database (MySQL), a server, operating system, application servers, and middleware, now there are cloud modules, APIs, and microservices taking their place.

3. Integration is simplified

Tools need to work together, and thanks to APIs and modular services, they can – and without a lot of hassle. Customer service platforms need to integrate with email and databases, automatically. Many of the new generation of software solutions not only work well together, they build on one another and can become incredibly powerful when paired up, for example, Salesforce's integrated SaaS.

4. Elasticity and affordable scalability

Cloud-based servers, databases, email, and data processing allow companies to rapidly scale up – something you can learn more in this Intro to Cloud Bursting article. Rather than provision more hardware and more time (and space) that it takes to set that hardware up, companies can purchase more space in the cloud on demand. This makes it easier to ramp up data processing. AWS really excels here, and is a top choice for companies like Upwork, Netflix, Adobe and Comcast have built their stacks with its cloud-based tools.

For areas like customer service, testing, analytics, and big data processing, modular components and services also rise to the occasion when demand spikes.

5. Flexibility and customization

The beauty of many of these platforms is that they come ready to use out the box – but with lots of room to tweak things to suit your needs. Because the parts are autonomous, you also have the flexibility to mix and match your choice of technologies – whether those are different programming languages or frameworks and databases that are particularly well-suited to certain apps or projects.

Another thing many organizations love is the ability to swap out one component for another without a lot of back-end reengineering. It is possible to replace parts in a monolith architecture, but for companies that need to get systems up and running fast – and anticipate a spike in growth or a lack of resources – modular components make it easy to swap out one for another. Rather than trying to adapt legacy technology for new

purposes, companies are beginning to build, deploy, and run applications in the cloud.

6. Real-time communication and collaboration

Everyone wants to stay connected and communicate – especially companies with distributed engineering teams. Apps that let companies communicate internally and share updates, information, and more are some of the most important parts of modern software stacks. Here's where a chat app like HipChat comes in, and other software like Atlassian's JIRA, Confluence, Google Apps, Trello, and Basecamp. Having tools like these helps keep everyone on the same page, no matter what time zone they're in.

7. Divvying up work between larger teams and distributed teams

By moving architectures to distributed systems, it's important to remember that the more complicated a system is, the more a team will have to keep up with a new set of challenges: things that come along with cloud-based systems like failures, eventual consistency, and monitoring. Moving away from the LAMP-style stack is as much a technical change as it is a cultural one; be sure you're engaging MEAN stack engineers and DevOps professionals who are skilled with this new breed of stack.

So what are the main platforms shaking up the stack landscape?

The Stackshare study dubbed this new generation of tech companies leaving LAMP behind as "GECS companies" – named for their predominant use of GitHub, Amazon EC2, and Slack, although there are many same-but-different tools like these three platforms.

Upwork has moved its stack to AWS, a shift that the Upwork engineering team is documenting on the Upwork blog. These new platforms offer startups and other businesses more democratization of options – with platforms, cloud-based servers, programming languages, and frameworks that can be combined to suit their specific needs.

- **JavaScript:** JavaScript is the biggest piece of the new, post-LAMP pie. Think of it as the replacement for the "P" (PHP) in LAMP. It's a front-end scripting language, but it's so much more – it's a stack-changer. JavaScript is powerful for both the front-end and back-end, thanks to Node.js, and is even outpacing some mobile technologies. Where stacks were once more varied between client and server, JavaScript is creating a more fluid, homogeneous stack, with a multitude of frameworks like Backbone, Express, Koa, Meteor, React, and Angular.
- **Ruby and Python** also dominate the new back-end stack, along with Node.js.

• **Amazon Web Services (AWS):** The AWS cloud-based suite of products is the new foundation for many organizations, offering everything from databases and developer tools to analytics, mobile and IoT support, and networking.

• **Computing platforms:** Amazon EC2, Heroku, and Microsoft Azure

• **Databases:** PostgreSQL, with some MongoDB and MySQL.

(6,970 symbols)

https://www.upwork.com/hiring/development/how-cloud-computing-is-changing-the-software-stack/

TEXT 36 BACK-END TECHNOLOGY: THE ROLE OF THE BACK-END WEB DEVELOPER

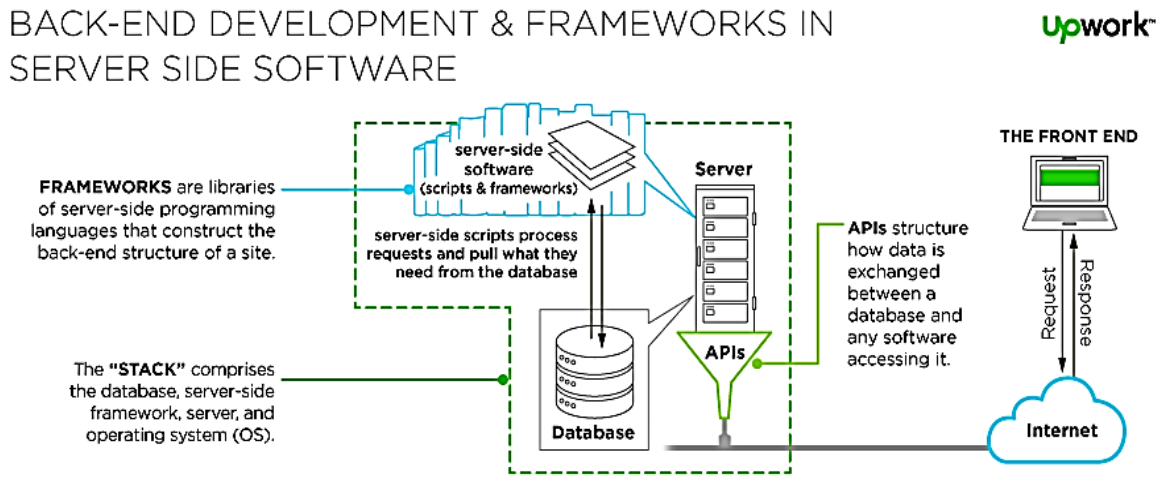
Your website or dynamic web application is a sum of layers – structure, design and content, and functionality. The technology and programming that "power" a site – what your end user doesn't see but what makes the site run – is called **the back end.** Consisting of the *server*, the *database*, and the *server-side applications*, it's the behind-the-scenes functionality – the brain of a site. This is the ecosystem of the database manager and the back-end developer.

Here's a look at the role of back-end programmers: their responsibilities, the environment they work in, the technologies they use, and related back-end skills.

Understanding the back end: adding function to form

The back end is the machine that runs a site – the user doesn't see it or directly interact with it as with client-side technology, but it's always running in the background, delivering smooth functionality, a desktop-like experience, and information from the database right into the browser.

Back-end development basics



- Back-end code adds utility to everything the front-end designer creates.

- The back end is a combination of a database and a software written in a server-side language, which are run on web servers, cloud-based servers, or a hybrid combination of both. A network's server set-up can vary, with the server-side workload divided up between various machines (e.g., a server dedicated to housing the database).
- This server-side application directly interacts with the database via an application programming interface (API), which pulls, saves, or changes data.
- The data are returned and converted into front-end code a user interacts with: filling out a form, creating a profile, shopping online, etc.
- In general, anything you see on a site is made possible by back-end code, which exists on, and is powered by, a server.

The back-end developers' toolbox

Back-end developers create and maintain the entire back-end function outlined above. The back-end developer takes finished front-end code and gives it working functionality – for instance, making values in a drop-down menu possible by building the infrastructure that pulls values from the database.

Other responsibilities of the back end can include

- Database creation, integration, and management – e.g., MySQL, SQLite, PostgreSQL, and MongoDB. SQLite is lightweight and fast, making it a very popular alternative to a larger MySQL driver.
- Using back-end frameworks to build server-side software, like Express.js
- Web Server technologies – e.g., J2EE, Apache, Nginx (popular for static content, like images, HTML or CSS files), and IIS
- Cloud computing integration – e.g., public cloud providers like Amazon Web Services, or private cloud environments
- Server-side programming languages – like Python, Perl, PHP, Ruby, and JavaScript, when implemented with the server-side development environment, Node.js
- Operating systems: Linux- and Unix-like operating systems, MacOS X, Windows Server
- Content management system (CMS) development, deployment, and maintenance
- API integration

- Security settings and hack prevents
- Reporting – generating analytics and statistics like system reports of server load, number of visitors, geography of visitors, etc.
- Backup and restore technologies for website's files and DB.

Server-side programming languages and frameworks

Back-end developers use an array of programming languages and frameworks when building server-side software. They may choose a framework to suit their working style or a site's specific requirements. They may also work with a language within a software stack. Popular back-end technology includes:

- **Ruby**: Great for building complicated logic on the database side of a site, Ruby bundles the back-end and database functionality that PHP and SQL can offer as a pair – it's great for startups, easy maintenance, and high-traffic demands. It requires the Ruby on Rails framework, which has vast libraries of code to streamline back-end development. *Ruby-powered sites: Hulu and Twitter*
- **Java:** A subset of the C language, Java comes with a huge ecosystem of add-on software components. At its core, Java is a variation of C++ with an easier learning curve; what's more, it's platform-independent thanks to the Java Virtual Machine. "Compile once, run anywhere" is its motto – and it's excellent for enterprise-level applications, high-traffic sites, and Android apps.
- **C#:** C# is an enhanced, second-generation version of the C language, one of the earliest back-end programming languages. C# is a general-purpose, object-oriented version specifically developed by Microsoft for the .NET Framework.
- **Python:** With fewer lines of code, the Python language is fast, making it ideal for getting things to market quickly. The emphasis is on readability and simplicity, so it's great for beginners. The oldest of the scripting languages, it is powerful and works well in object-oriented designs. *Python-powered sites: YouTube, Google*
- **PHP:** The most popular server-side language on the web, PHP is designed to pull and edit information in the database. It's most commonly bundled with databases written in the SQL language. PHP is unique in that it was built for the web, not adapted for it, and remains the most widely used language on the web. PHP has a number of modern frameworks as well.
- **Perl:** With 27 years of revisions and changes under its belt, Perl 5 is a high-level, general-purpose, interpreted language powerful for programming database integration with Oracle, Sybase, MySQL, and

more. It runs on more than 100 platforms and is – like Python and Ruby – object-oriented and open-source.

- **Erlang:** A general-purpose programming language, Erlang is also a concurrent language, which means several processes can run simultaneously on the language-level without external library support. It's used in the LYME and LYCE stacks, numerous CMS and databases, GitHub, and Goldman Sachs's platform, supporting its high-frequency trading requirements.
- **Node.js:** This breakthrough development environment, part of the JavaScript-powered MEAN stack, allows the front-end JavaScript language to be used in server-side applications with the Express.js framework.

Back-end software stacks

Depending on which "stack" you choose when building the back end, your back-end developer will need cross-component expertise.

What is a software stack? "Stacks" are just bundles of software for different aspects of your site's back end, combined for their compatibility and functionality to streamline development and deployment. The components include an operating system**,** a web server**,** a database**,** and server-side scripting language**.** You're not limited to the components in a stack – they're interchangeable based on your needs and customizable.

Two common stacks:

LAMP: Linux/Apache/MySQL/PHP

LAMP consists of free, open-source software components that work well for dynamic websites and applications. It's the most traditional stack model, with a few variations for different operating systems, servers, and database options. In the LAMP stack, PHP is interchangeable with the languages Python and Perl.

LAMP benefits: flexible, customizable, easy to develop, easy to deploy, secure, and comes with a huge support community since it's open source.

MEAN: MongoDB/Express.js/AngularJS/Node.js

MEAN is an all-JavaScript-powered replacement for the traditional LAMP stack. It's excellent for businesses looking to be agile and scalable, offering flexibility with the MongoDB document-based database and lots of features for building single- and multipage web applications. By using JavaScript across the front and back ends, developers working on the client side can easily understand the server-side code, which leads to greater productivity for your team.

MEAN benefits: single language used, supports the MVC pattern, uses JSON for data transfer, offers access to Node.js's JavaScript module library and the Express.js framework, is open source.

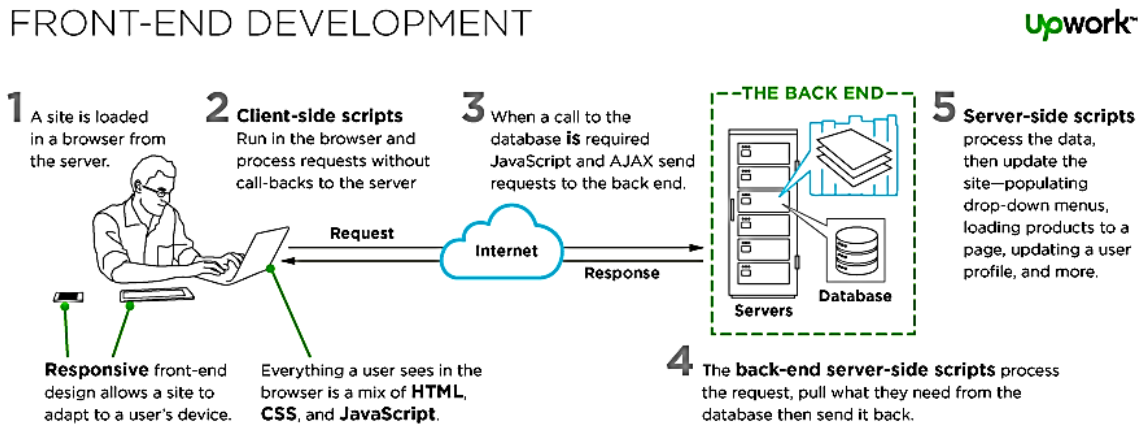
(6,382 symbols)

https://www.upwork.com/hiring/development/back-end-web-developer/

TEXT 37 FRONT-END WEB DEVELOPMENT: CLIENT-SIDE SCRIPTING & USER EXPERIENCE

Everything you see, click, and interact with on a website is the work of front-end web development. Client-side frameworks and scripting languages like JavaScript and AngularJS have made interactive websites possible. Here's a look at how this technology works in the scheme of a website, and some of the most popular scripts and frameworks you should know.

The Server vs. the Client



All websites run on three components: the server, the database, and the client. The client is simply the browser a person is using to view a site, and it's where client-side technology is unpacked and processed. *The server* is at a remote location anywhere in the world – housing data, running a site's back-end architecture, processing requests, and sending pages to the browser. *The client* is anywhere your users are viewing your site: mobile devices, laptops, or desktop computers. Server-side scripting is executed by a web server; client-side scripting is executed by a browser.

Client-end scripts are embedded in a website's HTML markup code, which is housed on the server in a language that's compatible with, or

compiled to communicate with, the browser. The browser temporarily downloads that code, and then, apart from the server, processes it. If it needs to request additional information in response to user clicks, mouse-overs, etc. (called "events"), a request is sent back to the server.

Client-side scripting is always evolving – it's growing simpler, more nimble, and easier to use. As a result, sites are faster, more efficient, and less work is left up to the server.

How Does Client-Side Scripting Work?

There is overlap between the two technologies as they work in tandem, but there are core differences. Server-side scripting works in the back end of a site, which the user doesn't see. It creates a scaffolding for the site to access its database, all the behind-the-scenes mechanics that organize and power a website. Client-side code, however, handles what the user *does* see.

- Scripts are embedded within and interact with the HTML of your site, selecting elements of it, then manipulating those elements to provide an interactive experience.
- Scripts interact with a cascading style sheet (CSS) file that styles the way the page looks.
- It dictates what work the server-side code is going to have to accomplish (where utility should be built around these front-end functions), and returns data that's pulled from the site in a way that's readable by the browser. For example: If there's a form for updating a profile, the back end is built to pull specific data from the database to populate that form, while front-end scripts populate the form with that information.
- Scripts put less stress on the server because they don't require processing on the server once they're downloaded, just when post-backs are made. "Post-backs" perform specific call-and-answers with the server-side code, and respond to the user immediately.

Client-Side Programming Languages & Frameworks

Now that you've got a broad view of what front-end technology is and does, here's a look at some of the most widely used scripting languages and front-end frameworks. Languages are almost always used in the context of their frameworks, which make quick work of complicated code with libraries of pre-packaged, shareable code, and lots of add-ons. Your developer may use one or a combination of these when building the front end of your site.

- HTML and CSS: These are the core building blocks of any site. HTML dictates a site's organization and content. CSS comprises the code for every graphic element – from backgrounds to fonts – that make up the look and feel of a website.
- JavaScript: JavaScript *is* client-side scripting. The most widely used client-side script – nearly every site's front end is a combination of JavaScript and HTML and CSS. JavaScript is fueled by an array of excellent frameworks that simplify it and give it more agility.

JavaScript Frameworks:

- AngularJS: An incredibly robust JavaScript framework for data-heavy sites
- JQuery, jQuery Mobile: A fast, small, JS object library that streamlines how JavaScript behaves across different browsers
- Node.js: A server-side platform that uses JavaScript, and is changing the way real-time applications can communicate with the server for faster response times and a more seamless user experience. It works with another JavaScript framework, Express.js, to build server-side Web applications.
- Bootstrap: A mobile-first framework that uses HTML, CSS, and JavaScript to facilitate rapid responsive app development
- React, for user interface design
- Express.js, Backbone.js, Ember.js, MeteorJS, and more
- TypeScript: A compile-to-JavaScript language that is a superset of JavaScript, created by Microsoft
- AJAX (JavaScript + XML) – a technology that allows specific parts of a site to be updated without a full-page refresh by asynchronously connecting to the database and pulling JSON- or XML-based chunks of data.
- VBScript & JScript are Microsoft's front-end scripts that run on the ASP.NET framework. JScript is Microsoft's reverse-engineered version of JavaScript.
- ActionScript, which creates animated interactive web applications for Adobe Flash Play
- Java (as "applets") snippets of back-end code that run independently with a run-time environment in the browser

Tip:

It's worth researching what browsers your primary audience is most likely to use, and what back-end scripts and APIs you're using, then decide on a script based on a stack or compatibility.

Client-Side Scripting Breakthroughs

An important breakthrough that changed the hard-and-fast rules for client side vs. server side? AJAX. The old standard was that server-side processing and page post-backs were used when the browser needed to interact with things on the server, like databases. AJAX, with its asynchronous calls to the server, can pull the data instantly and efficiently, without requiring a post-back. Read more about how it works in this AJAX Front-End Technology article.

Another major boost is jQuery, a fast, small, and feature-rich JavaScript library with an easy-to-use API that works across a multitude of browsers. Like code libraries do, jQuery changed the way that millions of people write JavaScript, simplifying a number of other client-side scripts like AJAX at the same time.

(5,176 symbols)

https://www.upwork.com/hiring/development/how-scripting-languages-work/

ЧАСТЬ 4: ГЛОССАРИЙ

A
Accomplish – выполнять
Agent-based – агентное моделирование.
Airspeed indicator – индикатор скорости воздушного потока
Altimeter – альтиметр/высотомер
Analyst – аналитик
API (Application Program Interface**) –** интерфейс прикладной программы
Applicable – применимый
Application silos – сервер приложения
Application – приложение
ASP (Active Server Pages) – активные серверные страницы (технология для создания Web-приложений)
Attitude indicator – авиагоризонт
Automation – автоматизация

B
Back up – резервное копирование.
Back-end development – разработка веб-приложений на стороне сервера
Baseline – базовая строка
Black-box testing – тестирование по стратегии черного ящика
Binary – двоичный

C
Clarity – четкость
Client-server database architecture – база данных архитектуры клиент / сервер
Cloud – «облако»
Code refactoring – улучшение кода
Coding standard – стандарт кодирования
Compass system – курсовая система
Concurrently – одновременно
Connectivity – возможности сетевого взаимодействия, сопряженность
Conscious intervention – сознательное вмешательство
Consistent – последовательный
Constraint – ограничение

Constructive cost model – моделирование стоимости разработки
Contribute – вносить вклад
CPU (Central Processing Unit) – центральный процессор, ЦПУ
CSS (Cascading Style Sheets) – каскадные таблицы стилей
Customer – заказчик
Customization – настройка

D
Data – данные.
Data store – хранение информации.
Database – база данных
Database model – модель БД.
DBMS (Database Management System) – система управления базой данных
Dedicated hardware resources – выделенные аппаратные ресурсы
Define – определять
Deliver – доставлять
Deploy – применять; использовать
Design flaws – ошибка проектирования
Design verification – верификация проекта
Developer – разработчик.
Development- разработка
Dimension – размерность
Distinguish – выделить
distribution – распределение
Dynamic Host Configuration Protocol (DHCP) – протокол динамической конфигурации узла
Dynamic IP addresses – динамический IP-адрес
Disposition – ликвидация

E
Editing tools – инструменити редактирования
Embash – длинное тире
Encryption – шифрование
Engagement – вовлечение, участие
Ensure – обеспечивать, убедиться
Enterprise – предприятие
Estimate – приблизительно подсчитывать
Evaluation – оценка
Extreme Programming – экстремальное программирование

F
Feasibility – возможность технической реализации
Feature – функция.
Feedback – обратная связь
File-processing systems – системы обработки файлов.
Flexibility – гибкость
Flight Director Systems – система командных пилотажных приборов, командно-пилотажная система
Flight instruments – пилотажно-навигационный прибор
Front-end development – разработка веб-приложений на стороне клиента
Functionality – функциональность

G
Gathering – сбор
Grant – предоставить
Grid computing – система распределения мощностей и систем хранения
Gyroscopic Systems – гироскопическая система

H
Hardware – аппаратное обеспечение.
Heading indicator – индикатор курсовых углов, указатель курса
Holistic – целостный
Host name – имя хоста (имя хост-компьютера)
HTML (HyperText Markup Language) – язык гипертекстовой разметки

I
Identification – идентификация
implementation – реализация
Incremental – возрастание
Independent – независимый
Integration – объединение
Intermediate – промежуточный
Intranet – Интранет
Intruder – нарушитель,злоумышленник
Involve – вовлекать
IP (Internet protocol) address – адрес сетевого протокола IP
Iteration – повторение
Iterative model – итеративная модель

J
Junkware-спам
Java [ˋdʒɑːvə] – Ява, язык программирования

K
Kernel – ядро

L
LAMP (Linux/Apache/MySQL/PHP**) software stack** – программный стек Linux/Apache/MySQL/PHP
Laptop – ноутбук
Layout – макет
Life-cycle – жизненный цикл (программного продукта)
Linear – линейный
Local area networks – локальные сети
Long-term – долгосрочный

M
Magnetic compass – магнитный компас
Mainframe – мейнфрейм
Maintain – поддерживать
Maintenance – поддержка
Malicious (program) – вредоносная (программа)
Malware – вредоносные программы
Management – управление.
Metaphor – метафора (программа)
Microcomputers – микрокомпьютер
Modem – модем (модулятор-демодулятор)
Modification – изменение
Modularization – деление на модули
Monitoring – контроль
Multitasking – многозадачность
Multi-tier – многоуровневый
Multi-user – многопользовательский

N

Navigational systems – навигационные системы
Network – сеть
No strings attached – без обязательств
Nondirectional Radio Beacon (NDB) – ненаправленный радиомаяк; приводная радиостанция
Normalization – нормализация

O

Object-oriented design – объектно-ориентированное проектирование
Object-oriented database systems – объектно-ориентированные СУБД
Object-oriented programming – объектно-ориентированное программирование.
Octet – октет (упорядоченный набор из 8 бит)
Ongoing – постоянный
Open source – открытым исходным кодом
Operation – работа, операция

P

Pair programming – парное программирование
Patch – патч
Persist – сохраняться
Phase – фаза
PHP (Personal Home Page) tools – Инструменты для создания персональных веб-страниц
Pipelining – конвейерный режим; конвейерная обработка
Pool – общий фонд
Preinstalled – предустановленный
Private network – частная (внутренняя) сеть
Procedure – процедура
Processing – обработка
Programmer – программист
Programming language – язык программирования
Prototype – прототип
Prototyping – прототипирование
Pseudo code – псевдокод; символический код

R

Ransomware – вредоносная хакерская программа (с требованием выкупа)
Recognize – распозновать
Recover – восстанавливать
Redistribute – перераспределять
Redundant – излишний
Refer – относиться
Relate – иметь отношение
Relational algebra – реляционная алгебра
Relational database – реляционная БД
Relational model – реляционная модель данных
Relationship – отношение
Release – выпуск
Relevant – соответствующий
Repository – хранилище
Requirement – требование
Requirement gathering – сбор требований
Resilient – упругий
Responsive Web Design – адаптивный веб-дизайн
Roadmap – схема, план действий
Robust – крепкий
Root user – привилегированный пользователь
Rootkit – руткит (набор программных средств для маскировки)
Router – маршрутизатор
Run – работать

S

Scope – объем
Scrutiny – внимательное изучение
Search Engine Optimization – оптимизация сайта в поисковых системах
Sequential – поэтапный, последовательный
Serviceable – обслуживаемый, полезный, пригодный
Single-user – однопользовательский.
Software – программное обеспечение.
Software design – проектирование программного обеспечения
Spiral model – спиральная модель
Spreadsheet – электронная таблица

Structured design – структурное проектирование
Structured programming – структурированное (структурное) программирование
Sub-domains – субдомен (поддомен)
Submit – отправить
Subnet – подсеть
Subsequent – последующий
Subset – подгруппа, разновидность
Subsystem – подсистема, совокупность модулей
Sync – синхронизация, синхронизировать
Systems Development Life Cycle (SDLC) – жизненный цикл разработки систем

T
T arrangement – расположение в форме буквы Т
TCP/IP protocol – протокол TCP/IP
Test case – тестовый пример
Threat – угроза
Top-level domain – домен высшего уровня
Transaction processing systems – системы обработки транзакций
Treat – рассматривать
Turn Indicator – курсоуказатель, гирополукомпас

U
Unauthorized – несанкционированный
Unwieldy – громоздкий
Updater – программа обновления
Upgrade – модернизировать
User – пользователь
User environment – среда пользователя
User Interfaces (UI) – пользовательский интерфейс
User-friendly – удобный, понятный
Utility – утилита
Utilized – использовать

V

V-model – V-модель (разработка через тестирование)
Validation – подтверждение соответствия
Verification – проверка полномочий, подтверждение
Vertical speed indicator – указатель скорости набора высоты
Very-High Frequency Omnidirectional Range (VOR) – всенаправленный азимутальный радиомаяк УКВ диапазона VOR
Vulnerability – уязвимость

W

Waterfall model – каскадная модель (водопад)
Web development – веб разработка
Web publishing – онлайн публикации
WEB-project – интернет-проекты
White-box testing – тестирование на основе стратегии белого ящика
Wi-Fi (Wireless Fidelity) – формат передачи цифровых данных по радиоканалам
Workgroup – рабочая группа

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Жидков, А. В. Понятие «научно-технический перевод» / А. В.Жидков // Science Time. – № 4 (4). – 2014. – С. 99–102.
2. Корухова, Л. В. Принципы отбора материала для подготовки студентами дополнительного домашнего чтения по английскому языку / Л. В. Корухова // Современные технологии обучения ино-странным языкам : Международная научно-практическая конференция (Ульяновск, 25 января 2012 года) : сборник научных трудов / отв. ред. Н. С. Шарафутдинова. – Ульяновск : УлГТУ, 2012. – С. 105–107.
3. Юсупова, Ш. Б. Некоторые сложности перевода английских технических терминов / Ш. Б. Юсупова // Молодой ученый. – 2015. – №4. – С. 808–811.
4. Tooley, Mike. Aircraft Digital Electronic and Computer Systems [Электронный ресурс] / M. Tooley. – 2nd edition. – Routledge, 2013. – Available at: http://www.readbook5.com/aircraft-digital-electronic-and-computer-systems/ (Accessed August 27, 2016).
5. What is 1C:Enterprise? [Электронный ресурс]. – Загл. с экрана. – Режим доступа: http://1c-dn.com/ (дата обращения 10.11.2016).
6. 3D printing.com [сайт] [Электронный ресурс]. – Режим доступа: http://3dprinting.com (дата обращения 10.11.2016).
7. Right now in computer [Электронный ресурс]. – Загл. с экрана. – Режим доступа: http://computer.howstuffworks.com (дата обращения 10.11.2016).
8. Oracle Help Center [Электронный ресурс]. – Загл. с экрана. – Режим доступа: http://docs.oracle.com (дата обращения 10.11.2016).
9. Goinglinux.com [сайт] [Электронный ресурс]. – Режим доступа: http://goinglinux.com (дата обращения 10.11.2016).
10. Vic Fay-Wolfe Fall2005 [Электронный ресурс]. – Загл. с экрана. – Режим доступа: http://homepage.cs.uri.edu/faculty/wolfe (дата обращения 10.11.2016).
11. ISTQB Exam Certification [Электронный ресурс]. – Режим доступа: http://istqbexamcertification.com (дата обращения 10.11.2016).
12. RonJeffries.com [сайт] [Электронный ресурс]. – Режим доступа: http://ronjeffries.com (дата обращения 10.11.2016).
13. SearchSQLserver [Электронный ресурс]. – Режим доступа: http://searchsqlserver.techtarget.com (дата обращения 10.11.2016).

14. The Tech Terms Computer Dictionary [Электронный ресурс]. – Режим доступа: http://techterms.com (дата обращения 10.11.2016).
15. Whatismyipaddress.com [сайт] [Электронный ресурс]. – Режим доступа: http://whatismyipaddress.com (дата обращения 10.11.2016).
16. Free essays, term papers, research paper and book report [Электронный ресурс]. – Режим доступа: http://www.123helpme.com (дата обращения 10.11.2016).
17. Free Online IT Tutorials and Internet Training [Электронный ресурс]. – Загл. с экрана. – Режим доступа: http://www.inetdaemon.com/tutorials (дата обращения 10.11.2016).
18. Kean University [Электронный ресурс]. – Режим доступа: http://www.kean.edu (дата обращения 10.11.2016).
19. Computer Support and Managed IT Services [Электронный ресурс]. – Режим доступа: http://www.nashnetworks.ca (дата обращения 10.11.2016).
20. Scientific American [Электронный ресурс]. – Режим доступа: http://www.scientificamerican.com (дата обращения 10.11.2016).
21. Tutorialspoint [сайт][Электронный ресурс]. – Режим доступа: http://www.tutorialspoint.com/ (дата обращения 10.11.2016).
22. Short Essays (Economics, Politics, Law and Business) [Электронный ресурс]. – Загл. с экрана. – Режим доступа: https://jennadoucet.wordpress.com (дата обращения 10.11.2016).
23. Gislounge [сайт] [Электронный ресурс]. – Режим доступа: https://www.gislounge.com (дата обращения 10.11.2016).
24. Quora – The best answer to any question [Электронный ресурс]. – Режим доступа: https://www.quora.com (дата обращения 10.11.2016).
25. The Guardian : Technology [Электронный ресурс]. – Режим доступа: https://www.theguardian.com/technology (дата обращения 10.11.2016).
26. Upwork – Hire Freelancers and Get Freelance Jobs Online [Электронный ресурс]. – Режим доступа: https://www.upwork.com (дата обращения 10.11.2016).
27. Мультитран [словарь] [Электронный ресурс]. – Режим доступа: www.multitran.ru (дата обращения 14.11.2016).

Учебное электронное издание

Go For IT English Reading
Учебное пособие
по английскому языку
для бакалавров 1−2 курса
факультета информационных систем
и технологий очной формы обучения

Составители: КОРУХОВА Людмила Владимировна
НОВОСЕЛЬЦЕВА Надежда Николаевна

ЭИ № 813. Объем данных 2,48 Мб. Заказ ЭИ 3.

Технический редактор Ю. С. Лесняк

ЛР № 020640 от 22.10.97.
Печатное издание
Подписано в печать 01.12.2016. Формат 60×84/16.
Усл. печ. л. 10,00. Тираж 120 экз. Заказ № 2.

Ульяновский государственный технический университет
432027, Ульяновск, Сев. Венец, 32.
ИПК «Венец» УлГТУ, 432027, Ульяновск, Сев. Венец, 32.
Тел.: (8422) 778-113
E-mail: venec@ulstu.ru
http://www.venec.ulstu.ru