ТЕХНОЛОГИИ В ОБЛАСТИ АППАРАТНОГО ОБЕСПЕЧЕНИЯ

Классы компьютеров, которые можно отнести к нанотехнологиям:

• Молекулярные компьютеры

• Квантовые компьютеры

• Нанокомпьютеры

Молекулярные компьютеры •

Молекулярные компьютеры — вычислительные системы, использующие вычислительные возможности молекул (преимущественно, органических). Молекулярными компьютерами используется идея вычислительных возможностей расположения атомов в пространстве.

Квантовые компьютеры •

Квантовый компьютер — вычислительное устройство, работающее на основе квантовой механики. Квантовый компьютер принципиально отличается от классических компьютеров, работающих на основе классической механики.

Куби́т (q-бит, кьюбит, кубит; от quantum bit) — квантовый разряд или наименьший элемент для хранения информации в квантовом компьютере.

Ограниченные (до 512 кубитов) квантовые компьютеры уже построены. Элементы квантовых компьютеров могут применяться для повышения эффективности вычислений на уже существующей приборной базе.

Нанокомпьютеры

Нанокомпьютер — вычислительное устройство на основе электронных (механических, биохимических, квантовых) технологий с размерами логических элементов порядка нескольких нанометров.

Сам компьютер, разрабатываемый на основе нанотехнологий, также имеет микроскопические размеры. На данный момент создан нанотранзистор — основа нанопроцессора.

Новый нанокомпьютер Физики собрали нанокомпьютер, нарушающий закон Мура •Группа ученых из Гарвардского университета построили сверхмалую вычислительную систему, функционирующую вне закона Мура. На сегодняшний день это самая плотная наноэлектронная система размером с нейрон человека и крайне низким электропотреблением. По мнению разработчиков, минипроцессоры такого типа могут использоваться для создания биомедицинских датчиков, микроимплантатов, мини-роботов, и, конечно, применяться в гибкой потребительской электронике.

Эволюция компьютерного аппаратного обеспечения

В истории развития компьютеров можно выделить четыре этапа, каждый из которых ознаменовался появлением компьютеров новых поколений (computer generations). Каждое поколение отличается технологией компонентов, ответственных за обработку данных. Также, на каждом этапе значительно увеличивалась скорость обработки данных и возможности их хранения при одновременном снижении цен (см. [Рис. 1.4](http://cdo.bseu.by/dl1/hardware/h1_3.htm#Рис. 1.4 Увеличение производительности и падение цен.)). Например, стоимость выполнения 100000 вычислений снизилась с нескольких долларов в 50-е годы до менее чем $0.025 в 80-е и примерно $0.00004 в 1995 году. Эти изменения в поколениях компьютерного аппаратного обеспечения сопровождались изменениями в программном обеспечении, что сделало компьютеры более мощными, недорогими и легкими в использовании.

|  |
| --- |
| Рис. 1.4 Увеличение производительности и падение цен. |
| http://cdo.bseu.by/dl1/hardware/img/1_4.gif |
| MIPS – миллион операций в секунду |

ПОКОЛЕНИЯ КОМПЬЮТЕРОВ

Компьютеры первого и второго поколений конструировались на основе электровакуумных ламп и транзисторов. Основой третьего и четвертого поколения стали полупроводниковые микросхемы.

Первое поколение: Электровакуумные лампы, 1946 – 1956

В первом поколении компьютеров для хранения и обработки информации применялись электровакуумные лампы. Лампы потребляли много энергии, были недолговечными и выделяли много тепла. Огромные в размерах, компьютеры первого поколения, имели маленькую память, их вычислительные способности были сильно ограничены. Поэтому применялись они для решения очень узкого круга научных и инженерных задач. Наибольший размер памяти этих компьютеров был 2000 байт (около 2 килобайт), а скорость обработки – 10000 инструкций в секунду. В качестве устройств внутренней памяти использовались вращающиеся магнитные барабаны, а для внешнего хранения данных применяли перфокарты. Такие задачи, как запуск программ и вывод на печать, координировались вручную.

Второе поколение: Транзисторы, 1957 – 1963

В компьютерах второго поколения в качестве устройств для хранения и обработки информации на смену вакуумным лампам пришли транзисторы. Транзисторы были более стабильны, чем лампы, они выделяли меньше тепла и потребляли меньше энергии. Однако каждый транзистор представлял собой отдельную деталь, которую нужно было впаять в печатную плату – медленный, трудоемкий процесс. На этом этапе в качестве первичных устройств хранения информации применялась технология памяти на магнитных сердечниках. Она состояла из маленьких (около 1 мм в диаметре) магнитных колец, которые поляризовались в двух направлениях, представляя таким образом бит данных. Эта память собиралась вручную, и поэтому была очень дорогой. Компьютеры второго поколения имели до 32 килобайт оперативной памяти, а скорость вычислений их была от 200000 до 300000 операций в секунду. Увеличение скорости обработки и количества памяти компьютеров второго поколения позволило использовать их для решения более широкого круга научных и бизнес-задач, таких как автоматическое выполнение платежных операций.

Третье поколение: Интегральные схемы, 1964 – 1979

Третье поколение компьютеров создавалось на основе интегральных схем (ИС), которые состояли из тысяч и тысяч крошечных транзисторов, помещенных внутрь микросхем. Память компьютеров расширилась до двух мегабайт, а скорость обработки возросла до 5 MIPS. Программное обеспечение компьютеров третьего поколения позволило использовать эти сложные машины людям, не имевшим специальной подготовки, что привело к усилению роли компьютеров в бизнесе.

Эпоха интегральных схем. С машинами третьего поколения связано значительное событие — разработка и внедрение визуальных устройств ввода-вывода алфавитно-цифровой и графической информации с помощью электронно-лучевых трубок — дисплеев, использование которых позволило достаточно просто реализовать возможности вариантного анализа. Дисплей принципиально изменил процесса ввода-вывода данных и упростил общение с компьютером.

Четвертое поколение: Сверх Большие Интегральные Схемы, 1980 – настоящее время

Четвертое поколение компьютеров зародилось в начале 80-х и существует по наши дни. Основой компьютеров этого поколения стали Сверхбольшие Интегральные Схемы (СБИС), в одном корпусе которых содержатся миллионы транзисторов. Цены снизились настолько, что компьютеры стали недорогими и нашли широкое применение в бизнесе и повседневной жизни. Мощь компьютера, занимавшего недавно большую комнату, переместилась в маленький корпус. Размеры оперативной памяти выросли до 7 и более гигабайт в больших машинах, применяемых для коммерческих расчетов; скорость обработки превысила 200 MIPS. В разделе 1.7 мы обсудим дальнейшие тенденции развития аппаратного обеспечения компьютеров.  
Технологии СБИС сделала возможным *микроминиатюризацию* – распространение компьютеров, которые столь малы, быстры и дешевы, что стали применяться повсеместно. К примеру, многие современные автомобили, стереосистемы, фото- и видеокамеры, игрушки, часы, даже устройства бытовой техники содержат микропроцессоры, управляющие работой этих устройств.

ЧТО ТАКОЕ МИКРОПРОЦЕССОР? ЧТО ТАКОЕ ЧИП?

Технология сверхбольших интегральных схем, содержащих миллионы транзисторов, позволила интегрировать в одну микросхему (чип) модули компьютерной памяти, логики и управления. Такие микросхемы получили название микропроцессоров (microprocessors). Микропроцессоры отличаются друг от друга своими характеристиками. Вы могли слышать о 8-битных, 16-битных или 32-битных процессорах. Этот показатель подразумевает разрядность (wordlength), или количество битов, которое машина может обработать за раз. 8-битный микропроцессор может обработать 8 бит, или 1 байт, за один *машинный цикл* (или *такт*). 32-битный микропроцессор может обработать 32 бита, или 4 байта, за один такт. Чем больше разрядность, тем выше скорость работы процессора.

Вторая важная характеристика процессора – *тактовая частота*. Каждое событие, происходящее в компьютере, должно быть последовательным, так чтобы один шаг в обработке логически следовал за другим. Модуль управления посылает в микропроцессор импульсы. Эти импульсы (такты) генерируются системным таймером; их частота измеряется в мегагерцах (megahertz), сокращенно МГц (MHz); 1Мгц – 1000000 тактов в секунду.

Первую регулярную статью тематической подборки под названием «Могут ли практики пренебрегать теорией, а теоретики – практикой» («Can Practitioners Neglect Theory and Theoreticians Neglect Practice?») представил ([Manfred Broy](mailto:broy@in.tum.de), Technical University of Munich, Germany).

Инженерия программного обеспечения по определению является частью практической дисциплины инженерии, и научные знания и интуиция в этой области не являются основными показателями успеха. Практики SE в основном фокусируются на своевременном построении и дальнейшем развитии программных систем с разумными затратами и качеством. Мерой успеха SE является та степень, в которой она обеспечивает понятия, методы и процессы, позволяющие разработчикам справляться с решением задачи развития программного обеспечения. Кроме собственно разработки и сопровождения программного обеспечения при решении этой задачи приходится иметь дело с анализом требований, архитектурой спецификаций, реализацией, интеграцией и валидацией.

Инженерные дисциплины должны основываться на научной теории и практике для обоснования корректности применяемых подходов и обеспечения научного подтверждения того, что используемые методы работают должным образом. К счастью, решению этой задачи способствуют многие научные методы, включая те из них, которые относятся к предметному анализу, эмпирическим и экспериментальным исследованиям, а также методы, заимствуемые из психологии экономики. Исследователи из области SE могут создавать, адаптировать и развивать эти научные методы для обеспечения более основательной поддержки своей дисциплины.

Автором статьи «Программное обеспечение с открытыми исходными текстами: связь с методами программной инженерии» («Open Source Software: Lessons from and for Software Engineering») является Брайана Фитцджеральд ([Brian Fitzgerald](mailto:bf@ul.ie), Lero—the Irish Software Engineering Research Centre).

Программное обеспечение с открытыми исходными текстами (Open Source Software, OSS) вызывает у людей противоречивые чувства. Сторонники этого направления утверждают, что OSS является бесплатным для пользователей высококачественным программным обеспечением, быстро производимым исключительно талантливыми разработчиками с очень низкими затратами. В то же время, критики характеризуют OSS как программное обеспечение, такое что его качество меняется от программы к программе, документация недостаточна или вовсе отсутствует, стабильность и надежность непредсказуемы, базовые правовые основы сомнительны. Все эти недостатки возникают вследствие хаотичного процесса разработки, которому абсолютно чужды фундаментальные принципы инженерии программного обеспечения и традиционный здравый смысл.

В своей статье автор излагает более сбалансированную точку зрения. С одной стороны, OSS не является той «серебряной пулей», которой его считают наиболее ярые приверженцы. С другой стороны, процесс разработки OSS не так радикально отклоняется от традиционной практики инженерии программного обеспечения, как это утверждают наиболее решительные противники OSS, и применение подхода OSS привело к созданию ряда программных продуктов, принесшим пользователям значительную пользу.

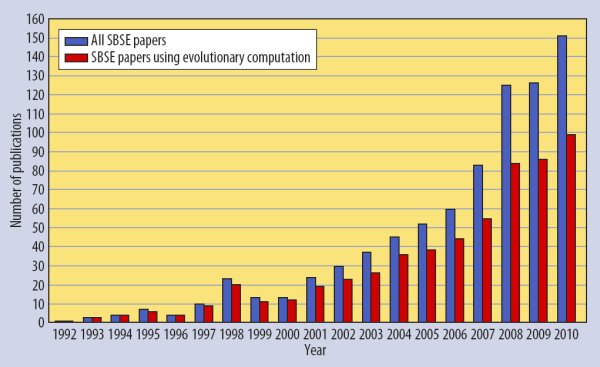
Следующая статья называется «Программная инженерия и эволюционные вычисления» («Software Engineering Meets Evolutionary Computation») и написана Марком Харманом ([Mark Harman](mailto:mark.harman@ucl.ac.uk), University College London).

Программное обеспечение эволюционирует. Этот факт был [осознан](http://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution) еще в самом начале истории программной инженерии. Хотя термин «эволюция программного обеспечения» используется по отношению к процессу, в ходе которого программные системы постоянно приспосабливаются к изменениям требований и среды функционирования, его нельзя считать чисто техническим; эволюция программного обеспечения имеет явную связь с классической дарвинской эволюцией.

Независимо от этого в компьютерных науках появился термин «эволюционные вычисления» (evolutionary computation) со строгим техническим смыслом: исследование алгоритмов, в которые закладываются критерии поиска в пространстве возможных решений с целью нахождения наилучших способов решения заданной проблемы. В сообществе эволюционных вычислений проводятся конференции и издаются журналы, посвященные разработке и применению эволюционных методов в автоматизированных процессах метаэвристической оптимизации.

В компьютерных науках эволюционные вычисления используются для оптимизации разработки артефактов и процессов в самых разнообразных инженерных областях. Однако, как это ни странно, до конца прошлого века сравнительно мало исследований посвящалось применению эволюционных вычислений (и других методов поисковой оптимизации) в области инженерии программного обеспечения. Это привело к образованию новой области исследований, называемой поисковой инженерией программного обеспечения (search-based software engineering, SBSE), которая посвящена применению методов поисковой оптимизации для решения проблем программной инженерии.

В последнее десятилетие исследователи применяли SBSE в ряде направлений программной инженерии, включая анализ требований, прогнозное моделирование, проектирование, тестирование и рефакторинг. Были разработаны многочисленные методы поисковой оптимизации. Описание некоторых из них содержится в [обзоре](http://www.dcs.kcl.ac.uk/technical-reports/papers/TR-09-03.pdf) автора статьи.

   
Рост числа публикаций по тематике SBSE, сопутствующий рост числа публикаций, посвященных применению эволюционных вычислений в инженерии программного обеспечения

Нет причин, по которым SBSE должна была бы опираться только на эволюционные вычисления; можно использовать и другие алгоритмы оптимизации, и они действительно используются. Например, в июне 2011 г. в 587 из 830 публикаций по тематике SBSE, собранных в [репозитории SBSE](http://crestweb.cs.ucl.ac.uk/resources/sbse_repository" \t "_blank), использовался, по крайней мере, один метод оптимизации. В 9,0% статей использовались какие-либо эволюционные алгоритмы (evolutionary algorithm), в 45,5% – генетические алгоритмы (genetic algorithm), в 13,5% – генетическое программирование (genetic programming), в 0,6% – эволюционные стратегии (evolution strategy), в 1,8% – метод роя частиц (particle swarm optimization), в 1,4% – алгоритмы оценки распределений (estimation of distribution algorithm) и в 0,8% – рассеянный поиск (scatter search). Однако эволюционные вычисления использовались в 71% всех статей по тематике SBSE, и это единственный метод оптимизации, применимый в любом направлении инженерии программного обеспечения.

Робин Лац ([Robyn Lutz](mailto:rlutz@iastate.edu), Iowa State University and Jet Propulsion Laboratory, California Institute of Technology) представил статью «Применение инженерии программного обеспечения в области исследований космического пространства» («Software Engineering for Space Exploration»).

Новые достижения в области построения автоматических космических аппаратов преобразуют исследования космического пространства. В космических аппаратах используется все больше программного обеспечения для поддержки более интеллектуального удаленного управления, более быстрого и точного обнаружения и устранения неисправностей, более совершенного сбора и анализа научных данных для лучшего понимания нашей вселенной.

В этом году NASA осуществило или подготовило космические экспедиции для составления карты гравитационного поля Луны (программа изучения гравитационного поля и внутреннего строения Луны, Gravity Recovery and Interior Laboratory, [GRAIL](http://science.nasa.gov/missions/grail)), исследования того, как сформировался Юпитер и другие гигантские планеты ([Juno](http://science1.nasa.gov/missions/juno" \t "_blank)), а также запустило программу Mars Science Laboratory ([MSL](http://mars.jpl.nasa.gov/msl)), в рамках которой на Марс будет доставлен автоматический марсоход Curiosity, предназначенный для исследования марсианской среды.

   
Эволюция марсоходов. Слева направо: один из двух марсоходов 2004 года – Spirit и Opportunity; марсоход Pathfinder 1997 года и марсоход Curiosity 2011 года (фотография NASA/JPL-Caltech)

Несмотря на все усилия инженеров, эти исследования являются опасными для автоматических космических аппаратов как вблизи Земли, так и в глубоком Космосе. Некоторые факторы внешней среды наносят вред бортовым программным системам. Например, перевертывание битов из-за радиации может привести к отказам или неверному функционированию программного обеспечения. Хотя некоторые неожиданности вполне приятны (например, космическому аппарату Voyager 2 засвидетельствовать взрыв сверхновой звезды), большинство непредвиденных событий или обстоятельств может привести к повреждению или разрушению систем.

Космические исследования поддерживаются разнообразным программным обеспечением. Полетное программное обеспечение управляет орбитальными космическими аппаратами, посадочными модулями и космическими зондами. Навигационное программное обеспечение управляет движением марсоходов. Бортовое программное обеспечение получает научные данные от планетоходов и приборов и передает их на Землю для дальнейшей обработки. Операционное программное обеспечение на Земле отслеживает космические экспедиции и управляет ими. Аналитическое программное обеспечение помогает ученым визуализировать и интерпретировать результаты и применять заново полученные знания для решения других проблем на Земле.

Инженерия программного обеспечения играет жизненно-важную роль для обеспечения инструментальных средств, методов и систематических подходов к разработке и поддержке многих видов программного обеспечения, требуемого в космических исследованиях. Применение разработки программного обеспечения на основе моделей (model-based software development), формальной верификации и разработки линеек продуктов (product-line development) обеспечивает соответствие возможностям программного обеспечения потребностям космических исследований, надежное и корректное функционирование космических аппаратов.

Автоматические космические аппараты, непилотируемые средства, которые управляются командами, посылаемыми с Земли или «зашитыми» в бортовые программы, являются авангардом космических исследований, поскольку они могут отправляться в места, недоступные для пилотируемых космических полетов. Поскольку многие автоматические космические аппараты выполняют далекие путешествия и должны нормально функционировать в течение долгого времени, они должны быть очень надежными. Для выявления возникающих аномалий и обеспечения должной реакции программное обеспечение таких аппаратов делается в высшей степени отказоустойчивым. Группы управления космическими аппаратами также могут во время полета изменять программное обеспечения для его адаптации к изменению потребностей или возникающим сбоям в работе компонентов аппарата.

В центре внимания автора находятся автоматические космические аппараты, разработанные в Лаборатории реактивных двигателей NASA (Jet Propulsion Laboratory. JPL), но программное обеспечение экспедиций по исследованию космоса разрабатывается и в других подразделения NASA, а также в[международных космических агентствах](http://en.wikipedia.org/wiki/List_of_space_agencies), коммерческих организациях и исследовательских институтах. Ограниченность бюджета, наличие различного уровня подготовки, стремление к распределению рисков и потенциальные преимущества совместной работы стимулируют сотрудничество нескольких организаций для решения задач, которые ни одна из них не могла бы решить в одиночку. Международное сотрудничество стало отличительной чертой области исследования космоса.

Статья «Инженерия программного обеспечения, сервис-ориентированные архитектуры и “облачные” вычисления» («Software Engineering Meets Services and Cloud Computing») написана Стивеном Яу и Хо Эном ([Stephen S. Yau](mailto:yau@asu.edu), [Ho G. An](mailto:ho.an@asu.edu), Arizona State University).

Сервис-ориентированные и «облачные» вычисления привлекают большое внимание индустрии и академии, поскольку эти подходы позволяют быстро разрабатывать масштабные распределенные приложения в таких областях, как совместные исследования и разработки, электронный бизнес, здравоохранение, приложения с использованием grid, корпоративные компьютерные инфраструктуры, военные приложения и национальная безопасность. Эти парадигмы компьютинга облегчают и удешевляют создание как простого коммерческого программного обеспечения, так и сложных критически важных приложений.

В обеих парадигмах используются парадигмы аутсорсинга ресурсов и передачи информационных технологий в ведение поставщиков услуг, но в делается упор на разные аспекты инженерии программного обеспечения. В сервис-ориентированном компьютинге на переднем плане находится архитектурный дизайн, что позволяет разрабатывать приложения путем обнаружения и композиции сервисов. «Облачные» вычмсления ориентируются на эффективную доставку сервисов за счет гибкой и масштабируемой виртуализации ресурсов и балансировки нагрузки.

Сервис-ориентированная инженерия программного обеспечения (service-oriented software engineering,[SOSE](http://www.public.asu.edu/~ychen10/book/socwsi.html#_Toc296290307)) объединяет лучшие черты этих парадигм. Изначально SOSE основывалась на сервис-ориентированном компьютинге, но впоследствии стала затрагивать и «облачный» компьютинг. В SOSE сервис-ориентированная архитектура обеспечивает архитектурный стиль, стандартные протоколы и интерфейсы, требуемые для разработки приложений, а «облачная» инфраструктура используется для доставки пользователям требуемых сервисов за счет виртуализации и применения пулов ресурсов. Соединение сервис-ориентированного и «облачного» компьютинга в инфраструктуре программной инженерии может помочь разработчикам и поставщикам сервисов справиться с проблемами каждой из этих парадигм.

Хотя в принципе SOSE является перспективной, для ее реализации потребуются дополнительные исследования в области инженерии программного обеспечения для решения проблем безопасности и качества обслуживания, возникающих в сервис-ориентированном и «облачном» компьютинге.

Последняя статья тематической подборки написана Дэвидом Лорджем Парнасом ([David Lorge Parnas](mailto:parnas@mcmaster.ca), Middle Road Software) и называется «Инженерия программного обеспечения – без вести пропавшая: личная точка зрения» («Software Engineering – Missing in Action: A Personal Perspective»).

Профессиональная карьера автора началась примерно в то же время, когда начал использоваться термин «инженерия программного обеспечения». До этого люди использовали термины типа «программирование» или «разработка программного обеспечения», когда говорили о работе, которую сегодня назвали бы программной инженерией.

Уже в 1960-х гг. программное обеспечение начинало становиться основной проблемой. Было принято считать, что аппаратура компьютера готова, и работает, когда программное обеспечение является еще неполным и ненадежным. Разработка программного обеспечения обычно не укладывалась в выделенный бюджет, нарушались плановые графики, а конечный результат не отвечал ранее установленным требованиям.

Многие разработчики программного обеспечения начали понимать, что они делают не совсем то, чему их учили. Они готовились стать учеными или математиками и вносить свой вклад в общечеловеческое знание. В действительности же им приходилось производить продукты, которыми должны были пользоваться другие люди.

Основатели движения «инженерии программного обеспечения» отмечали, что Инженеров учат хорошо справляться со своей работой. В отличие от этого, разработчиков программного обеспечения учили совсем не тому, над чем им пришлось работать. У инженеров, несмотря на все их несовершенство, не было столько проблемных проектов, сколько их встречалось у разработчиков программного обеспечения. И основатели движения надеялись, что многих проблем разработки программного обеспечения удалось бы избежать, если бы была сформирована соответствующая новая инженерная дисциплина.

Однако, по мнению автора, хотя по поводу инженерии программного обеспечения было написано множество статей и было предложено много интересных идей, исследователям и практикам так и не удалось создать новую инженерную дисциплину, ориентированную на создание программных систем. Статьи этого номера, посвященные связям программной инженерии с другими областями исследований, только подтверждают эту точку зрения.

* Если до сих пор пишутся статьи, в которых утверждается, что разработчики программного обеспечения нуждаются в некоторой теории, а не ведутся разговоры о каких-либо конкретных областях теории, которые должны знать инженеры программного обеспечения, то, значит, профессия программного инженера еще не сформировалась.
* Если обнаруживается, что в отдельных прикладных областях имеется общие проблемы, которые решаются на основе их собственной терминологии и специализированных методов, то, значит, профессия программного инженера еще не сформирована.
* Если область программной инженерии завоевывает популярность за счет причудливых изобретений с замысловатыми аббревиатурами, которые вызывают шквал интереса, а потом исчезают из поля зрения, то, значит, профессия программного инженера еще не сформирована.
* Если продолжать относиться к программной инженерии как к области исследований «наугад», а не как к организованной профессии, то теряется из виду исходная цель этого движения.

Родоначальники инженерии программного обеспечения были прозорливы. Как кажется, они предвидели появление тяжкой зависимости от программного обеспечения. Должно быть, они понимали, что программная инженерия должна стать одной из многих сложившихся инженерных профессий. К сожалению, их последователи недооценили трудности достижения этой цели и потеряли ее из виду.

Если мы хотим образовать инженерную дисциплину, специализирующуюся на разработке программных систем, то, прежде всего, нужно придти к согласию о требуемой для этого базовой совокупности знаний. До сих пор все люди, пытавшиеся сформировать подобную совокупность знаний, стремились к слишком широкому охвату. Они старались собрать все мнения и факты о разработке программного обеспечения вместо того, чтобы выявить небольшое ядро надежных знаний, которыми должны обладать все программные инженеры.

По опыту автора статьи, сформировать такое ядро совсем не просто. При недостаточной бдительности это ядро будет постоянно расширяться и, следовательно, утрачивать свою актуальность из-за слишком большого размера.

Вне тематической подборки в октябрьском номере опубликована статья «Управление питанием дисплея с выявлением намерений пользователей» («Display Power Management That Detects User Intent»), авторами которой являются Джей Мин Ким, Миньен Ким, Джунго Конг, Хиюнг Беом Джанг, Сун Ву Чунг ([Jae Min Kim](mailto:joist@korea.ac.kr), [Minyong Kim](mailto:mkim05@korea.ac.kr), [Joonho Kong](mailto:luisfigo77@korea.ac.kr), [Hyung Beom Jang](mailto:kuphy01@korea.ac.kr), [Sung Woo Chung](mailto:swchung@korea.ac.kr), Korea University).

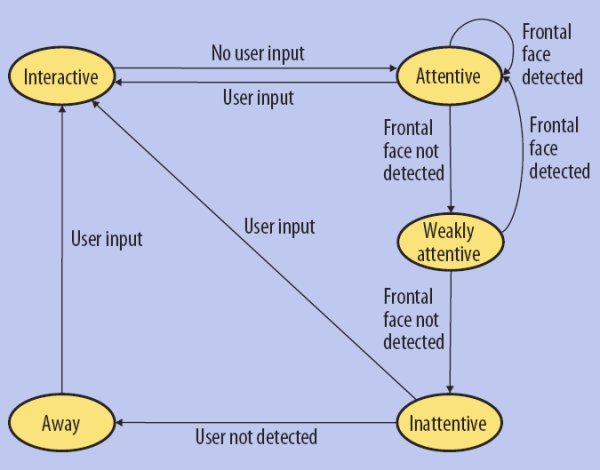
Расширение областей применения компьютеров приводит к тому, что все чаще люди используют лаптопы, а не настольные компьютеры. Соответственно, возрастает и актуальность проблемы продления времени работы аккумуляторных батарей на счет продуманного управления питанием дисплеев (display power management, DPM).

Хотя энергопотребление дисплея различно у разных лаптопов, оно составляет около 30% от общего энергопотребления устройства. Эта процентная доля выглядит справедливой, когда пользователям требуются высококачественные изображения, видео и т.д., но слишком часто высокий уровень энергопотребления сохраняется даже в тех случаях, когда пользователь на экран не смотрит. Вопреки распространенному мнению, скринсейверы не слишком способствуют снижению уровня энергопотребления. Вычислительные накладные расходы на запуск скринсейвера могут даже повысить общее энергопотребление системы.

Экономии энергии ничто не способствует больше, чем простое выключение дисплея. Поэтому очевидным направлением совершенствования DPM является создание механизма, который может выявлять намерения пользователя и выключать дисплей не только когда пользователь отсутствует, но также и тогда, когда дисплей не используется.

Решению этой проблемы посвящалось несколько исследований. В типичных операционных систем обеспечиваются DPM с таймаутом, выключающие дисплей при длительном отсутствии ввода. В некоторых работах предлагались DPM с более тонкими механизмами выявления намерений пользователей. Такие механизмы основывались, например, на использовании ультразвука для обнаружения присутствия пользователя. Однако большая часть подобных схем направлена, прежде всего, на снижение энергопотребления в тех случаях, когда пользователь отходит от дисплея.

Авторы статьи разработали схему, улучшающую выявление намерений пользователей за счет обнаружения состояний пользовательского присутствия за дисплеем на основе изображений, получаемых с Web-камер. Это позволяет выявить, например, внимателен ли пользователи или же невнимателен. Чтобы избежать ложного выявления намерений, было введено промежуточное состояние «слабое внимание», приводящее в тому, что алгоритм анализа изображения лица пользователя через несколько секунд анализирует новое изображение.

   
Диаграмма смены состояний предложенной схемы DPM. Использование клавиатуры или мыши переводит механизм в активное состояние. Web-камера позволяет установить, смотрит пользователь на дисплей, не смотрит или вовсе отсутствует. Состояние слабого внимания не соответствует какому-либо физическому состоянию пользователя. Оно заставляет алгоритм анализа изображения лица пользователя повторно проверить, смотрит ли пользователь на экран.

Авторы экспериментально установили, что их схема поддерживает лаптоп в режиме пониженного потребления энергии в течение примерно 50% времени его использования, а другие DPM достигают этого только в течение 30-40% времени. При этом эффекты, вызывающие раздражение пользователей, возникают не чаще одного раза в час.