М.Н. Хохлова

Робот по программированию

ЗАКОНОМЕРНОСТЬ ЭВОЛЮЦИИ МЕТОДОВ
РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

АБСТРАКТ

Цифровая трансформация экономики и эффективность применения любой вычислительной техники определяется **функциональным** программным обеспечением (ПО). За красивой объединяющей метафорой *«интернет вещей» (Internet of Things, IoT)*, *«интернет всего» (Internet of Everything)*, «умные вещи» *(Smart Things)* и т.п. сегодня примитивно скрываются миллионы «лоскутных» автономных программ, разработанных миллионами программистов.

Для анализа и оценки эволюции программирования и определения тенденций будущего развития рассмотрены ряд ключевых аспектов:

- **Объект управления,** какие требования он формирует к современному ПО.
- **Языки программирования**, эволюция их развития, что дальше...
- **Программа,** форма представления результата: текст и исполняемый код.
- **Программирование**, как повысить эффективность жизненного цикла ПО.
- **Программист.** Человеческий фактор, «сильное» и «слабое» звено ПО.
- **Кибербезопасность**. Уязвимости, новые принципы и подходы.

В данной работе изложена новая парадигма радикального концептуального изменения традиционного неторопливого развития программирования. Понимая острую потребность в создании нового класса адаптивных сложных информационных систем и платформ, адекватных динамике изменения требований природоподобных объектов управления, впервые создан **РОБОТ (АВТОМАТ/СТАНОК,..) ПО ПРОГРАММИРОВАНИЮ - G3AP.**

То есть предлагается вообще *не писать тексты программ* и *изъять программиста* из процесса распределённого коллективного (в реальном времени) создания сложных динамически развиваемых «живых» функциональных информационных систем и платформ. Принципиально изменяется традиционный многоэтапный жизненный цикл информационных систем. Наряду с рядом других **GGG**-технологий робот по программированию **G3AP** знаменует начало конца информационной эпохи. (Подробнее можно ознакомиться на интернет-ресурсе: http://www.viphmn.ru в книгах «КОНЕЦ ИНФОРМАЦИОННОГО ОБЩЕСТВА. НОВЫЙ РЕНЕССАНС», «ТЕОРИЯ ЭВОЛЮЦИОННОГО МОДЕЛИРОВАНИЯ», «ЭРА ГАРМОГЕНЕЗА», «Сетевые Мир и Война», «Глобальные вызовы и решения» и других)

Ключевые слова: *языки программирования, программист, машинные коды, языки высокого уровня, структурное программирование, специализированные языки, визуальное программирование, объектно-ориентированное программирование, семантическая интероперабельность, управление изменениями требований, робот по программированию, автоматическое программирование, ЭВМ, информационная система, программное обеспечение, «интернет вещей» (Internet of Things, IoT) «интернет всего» (Internet of Everything), сетецентрическая система, жизненный цикл,…*

ВВЕДЕНИЕ

В цифровой экономике целесообразность и эффективность применения любой **вычислительной техники/ЭВМ** *(которая является составной частью центров обработки данных, электронного гаджета, суперкомпьютера, стиральной машины, автомобиля, станка, технологического и бизнес процесса, самолёта, космического корабля, государства,..)* **определяется функциональным программным обеспечением** (ПО, soft-ом, прикладной (управляющей) программой, цифровым (электронным) сервисом, приложением,..).

В мире ежеминутно хаотически людьми (по профессии программист) создается и изменяется миллионы программ различных видов, которые ради «продажного» маркетинга как только не называют и как только ими не пугают (например, «искусственный интеллект»), окружая завесой тайны, чтобы меньше понимали, а больше «верили». Со всем этим хайпом ПО давно пора системно разобраться, сформировать из представленного «винегрета» понятий их комплексную взаимоувязанную архитектуру – единую целостную карту ПО:

АСУ-автоматизированные системы управления, информационные системы, «интернет вещей» (Internet of Things, IoT), «интернет всего» (Internet of Everything), базы данных, базы знаний, big data, гео-информационные системы, поисковые системы, электронная почта, сайты, блоги, социальные сети, онтологии, семантические сети, «умные» вещи, телемедицина, навигаторы, гибридные и «облачные» технологии, интегрированные системы и экосистемы, Системы Систем (System of Systems, SoS), сервис-ориентированная архитектура (Service-Oriented Architecture, SOA), электронные игры, «машинный интеллект», «интеллектуальные» системы распознавания текстов, образов, звуков, запахов,.., «самообучающиеся» компьютеры (Machine Intelligence, deep learning, machine learning, OpenAI, Deep Mind), библиотеки алгоритмов, криптовалюты, блокчейн, нейронные сети, мульти-агенты, грид-системы, сервисы, электронное государство, имитационное моделирование, нейронет, «искусственный интеллект»,..

Программистов сегодня зовут новым пролетариатом информационного общества. И вот уже который год все международные экспертные компании предрекают высокий рост потребности в ИТ-специалистах.

В списках ФОРБС именно «цифровые виртуалы» занимают совокупные лидирующие позиции во множестве рейтингов вполне материального благосостояния.

Учёные всего мира непрерывно «изобретают» методологии, стандарты и регламенты, которые смогли бы обеспечить продуктивность, повторяемость, и предсказуемость процесса программирования. Что только они не делают:

пытаются систематизировать и формализовать опыт о «правильном» пути создания программ, пишут локальные и международные стандарты, снижают сложность систем путем абстракции, фрагментирования и разграничения полномочий, применяют методы проектного управления, контролинга и программной инженерии,..

Но всегда получается идентичный результат - сложность предлагаемой методологии сравнима со сложностью самого программного продукта, а это влечёт неоправданное увеличение сроков и стоимости разработок информационных систем.

До настоящего времени все убеждены, что **программирование** – написание специального текста команд для вычислительных устройств, является настолько сложной задачей (*смесью науки и искусства*), что трудно поддаётся автоматизации.

Поэтому на всех этапах эволюции программирования особые усилия направлялись на совершенствование и повышение *уровня интеллектуальности интерфейса* между программистом и ЭВМ - на **языки программирования**.

Немного истории. Считается, что дочь лорда Байрона - леди Ада Лавлейс была первым программистом механического компьютера, созданного в начале XIX века английским ученым Чарльзом Беббиджом. Её программа решала уравнение Бернулли, выражающее закон сохранения энергии движущейся жидкости. Позднее её имя «Ada» было присвоено одному из языков программирования, который являлся базовым для министерства обороны США.

Не должно быть удивительно, что в столь «мужской профессии» как программирование, именно женщина первой решила концептуально и радикально изменить традиционную парадигму многолетнего неторопливого развития программирования, впервые создав

РОБОТА (АВТОМАТ/СТАНОК,..) ПО ПРОГРАММИРОВАНИЮ - G3AP.

То есть предлагается вообще ***не писать тексты программ*** **и** ***изъять программиста*** из процесса распределённого коллективного создания сложных динамически развиваемых информационных систем и платформ.

Кроме того, ***концептуально изменен традиционный многоэтапный (более 20 этапов) итерационный жизненный цикл информационных систем на 2х- этапный*** – **G3LC**.

Настоящая статья посвящена предпосылкам появления и описанию основных принципов работы инновационной технологии автоматического программирования **G3AP** - робота по программированию, где

- **G3** – GGG: Global Gnoseology Graph/Глобальный Гносеологический Граф,
- **AP –** Automatic Programming/Автоматическое Программирование.

В настоящее время сформирована и динамично эволюционно развивается единая взаимоувязанная СЕТЕВАЯ МОДЕЛЬ АРХИТЕКТУРЫ ПО (целостная динамическая карта всех видов существующего и необходимого программного обеспечения).

Модель создается для объективного исследования, анализа и научно-обоснованного прогноза развития и дальнейшего использования различных видов ПО (научно-исследовательский проект «Ноев ковчег ПО», что мы возьмем в будущее, а что безнадежно устареет).

Все предлагаемые в данной статье инновационные информационные технологии (инструменты и платформы) имеют практическую реализацию.

Наряду с рядом других **GGG**-технологий **G3AP** знаменует начало конца информационной эпохи.

Однако всё по порядку.

ЭВОЛЮЦИЯ ПРОГРАММИРОВАНИЯ

Программирование - процесс создания, сопровождения и развития компьютерных программ (программного обеспечения, ПО).

Инженерно-техническая дисциплина «программная инженерия» включает следующие основные этапы:

анализ и постановка задачи, проектирование архитектуры программы, модель предметной области, инфо--логическое и дата-логическое моделирование, выработка системных соглашений, построение алгоритмов, разработка структур данных, написание текстов программ, сборка программы, отладка и тестирование программы (испытания программы), документирование, настройка (конфигурирование), доработка, сопровождение, развитие, интеграция с другими программами,… утилизация.

На эволюцию программирования как научно-технологическую дисциплину влияет множество факторов, в том числе следующие основные:

- развитие вычислительной техники,
- развитие средств коммуникации,
- увеличение разнообразия, повышение сложности и динамики изменения задач, решаемых с помощью ЭВМ,
- повышение требований к простоте освоения и применения («юзабилити») системного и прикладного программного обеспечения,
- повышение качества программ, стремление разработчиков к созданию более совершенного результата,
- опыт поддержки, развития, интеграции унаследованных программ,
- повышение эффективности процесса производства программ и других этапов жизненного цикла информационных систем вплоть до утилизации,
- экономика процесса программирования,
- человеческий фактор - человеку удобнее описывать моделируемый в программе объект и процесс в терминах предметной области, а не специализированным языком символов и цифр,
- обеспечение информационной (в том числе кибер) безопасности,
- и многие другие.

Чтобы провести анализ и оценить основные стадии **эволюции программирования**, а так же определить тенденции его будущего развития предлагается рассмотреть шесть ключевых аспектов:

- **Объект управления**, какие требования он формирует к современному ПО.
- **Языки программирования**, эволюция их развития.
- **Программа**, форма представления ПО как результата программирования.
- **Программирование,** эффективность управления жизненным циклом ПО.
- **Программист,** человеческий фактор, роль «сильного» и «слабого» звена в создании сложных динамических информационных систем.
- **Кибербезопасность**, как её обеспечить для сложных динамических программных систем.

Рассмотрим детальнее все эти аспекты: актуализируем особенности и проблемы для **системной формализации задач и требований к ИТ-прорыву - концептуально новым подходам и решениям.**

Далее каждому направлению будет посвящена отдельная глава.

ОБЪЕКТ УПРАВЛЕНИЯ. ОСНОВНЫЕ ТРЕБОВАНИЯ

Основными принципами в общей традиционной теории систем являются *редукционизм* (сведение системы к ее составным частям), *холизм* (рассмотрение системы как целого) и *иерархия* системы и ее подсистем.

Тысячелетиями рядом с природными «живыми» системами человечество создавало **антропогенные искусственные системы** – здания, мосты, танки, самолеты, утюги, электростанции, заводы, 3D-принтеры, компьютеры, космические корабли, роботов,..

Все искусственные системы до настоящего времени люди создавали механистично и креационно (собирая целое или трансформируя его с помощью отдельных элементов), хотя и часто использовали биологические и природо-подобные декоративные понятия «наследование», «выращивание», «генезис», «эволюция», «адаптивность», «обучение», «самоорганизация»,...

Механистические (редукционные) системы отличаются от живых тем, что их можно разобрать и собрать без потери свойств.

Во всех учебниках мира сложные искусственные системы (проекты) предлагается создавать, используя следующие именно механистические «проектные» приемы и «научный подход»:

- определить цель, задачи,
- выделить создаваемую систему: *объекты и процессы* управления,
- декомпозировать систему на подсистемы,
- описать границы системы, оценить воздействия внешней среды,

- определить основные значащие характеристики и методы,
- и так далее…

Эти подходы традиционно используются и при создании сложных информационных систем и платформ, в том числе для глобальной цифровой трансформации экономики.

Программы до сих пор рассматривались как аналог любого искусственного устройства, созданного человеком для достижения поставленных целей.

Увеличить >>>

Задачи могут быть простыми и сложными. *Уровень сложности устройства(программы) должен был соответствовать сложности решаемой задачи*.

Данный исторически устоявшийся **подход будет концептуально изменен**.

Для создания сложного программного продукта все архитекторы во всем мире «искусственно» и каждый по-своему разделяют его на более простые составные части, а каждую часть, в свою очередь, делят еще и еще, пока не смогут эффективно описать каждый выделенный *элемент (подсистему, модуль, блок, сервис, приложение…*).

Такая последовательная многократная иерархическая декомпозиция применялась во всех направлениях деятельности человека для достижения следующих основных осознаваемых целей:

- уменьшения сложности системы и формирования самой возможности решения поставленной комплексной задачи,
- обеспечения ведения параллельной работы над частями сложной системы, для сокращения времени создания.
- реализации возможности сборки (интеграции, комплексирования) сложного объекта из формализованных унифицированных распределённо производимых модулей.

Каждый модуль в этой функциональной декомпозиции сложной системы представляет собой «*черный ящик*» с входом и выходом. Миллионы программных продуктов, разрабатываемых до настоящего времени во всех странах мира, имеют именно такую традиционную модульную архитектуру.

Но растёт потребность в решении все более сложных, динамично изменяющихся и развивающихся, взаимосвязанных, разно-дисциплинарных задач. Программы становятся сложнее, больше по объему и количеству «лоскутных» модулей - наборов «черных ящиков».

Интеграция подсистем в единое целое, попытки обеспечить их «бесшовное» **семантическое взаимодействие (интероперабельность)** стали отдельной дисциплиной технологической деятельности и программной инженерии.

Более детально с проблемами интеграции вы можете ознакомиться в статье «SOA. СМЭВ. ЭЛЕКТРОННЫЙ ОБМЕН ИЛИ ОБМАН. ПРОБЛЕМЫ ИНТЕГРАЦИИ»

Сегодня все приходят к пониманию острой потребности в создании **адаптивных** (приспосабливающихся) информационных систем, **адекватных** динамике изменения требований сложных «живых» объектов и процессов реального мира.

Многие уже осознают, что оперативная согласованность и синхронизация изменений данных, параметров и структур, методов обработки и форм визуализации в отдельных программных частях (модулях) сложной системы не может быть обеспечена в принципе.

Доказана теорема:

семантическая интероперабельность трёх и более динамических программных приложений не достижима.

Понятными, популярными и общепринятыми иллюстрациями модульных механистических подходов являются наборы *лего, пазлов, кубиков...*, из которых как бы «строятся» сложные комплексные программы.

А теперь представьте, что будет со «строительством» единого целостного «здания», если у каждого «кубика» в разное время по-своему и непрерывно должны изменяться форма, содержание и поведение из-за динамичного изменения требований к ним, например, изменились автоматизируемые объекты и процессы управления, выявлены ошибки или необходима дальнейшая оптимизация.

Катастрофа!

Модернизация миллионов модулей фрагментарных информационных систем всех компаний мира: SAP, ORACLE, GOOGLE, EADS, THALES, LOCKHEED MARTIN, IBM, Microsoft, SAS и других – находится в **концептуальном тупике**, **создает угрозу национальной и глобальной безопасности.**

Например, приведем краткий перечень видов программных модулей социально-экономических систем управления:

MRP - Material Requirements Planning, ERP - Enterprise Resource Planning, AMHS - Automated Material Handling System, APC - Advanced Process Control, APS - Advanced Planning & Scheduling, BPM - Business Process Management, CMM - Collaborative Manufacturing Management, CPAS -

Collaborative Process Automation System, CPM - Collaborative Production Management, VMI - Vendor Managed Inventory, CPS - Collaborative Planning & Scheduling, CRM - Customer Relationship Management, CSR - Customer Service Representative, EAM - Enterprise Asset Management, EMS - Electronic Manufacturing Services, LIMS - Laboratory Information Management System, WMS - Warehouse Management System, NPI - New Product Introduction, OpX - Operational Excellence, PAM - Plant Asset Management, PDM - Plant Data Management, PLM - Product Lifecycle Management, PSC - Plant Services Connector, PSM - Product Service Management, SBA - Service-Based Architecture, SBI - Service-Based Infrastructure, SCM - Supply Chain Management, SCPM - Supply Chain Process Management, SEM - Strategic Enterprise Management, SFA - Sales Force Automation, SRM - Supplier Relationship Management, TMS - Transportation Management System, KM - Knowledge Management,…

Как эти системы и соответствующие им модули программного обеспечения семантически взаимоувязаны, интегрированы? Слабо или никак.

А комплексный объект, где они должны вместе работать один и един – например, обычная транснациональная корпорация.

Могут ли они согласованно развиваться? НЕТ, НИКОГДА.

Кроме того, «винегрет» различных информационных систем, используемых на одном вычислительном устройстве, представляет собой «решето» уязвимостей (которые определяются «самым слабым звеном» - модулем), что только усугубляет проблемы обеспечения комплексной **кибербезопасности**.

Сегодня, наконец, приходит понимание, что нужны **принципиально новые подходы** для объектов и процессов, которые обладают следующими основными характеристиками:

- **сложность** – объекты состоят из большого числа разнотипных элементов и взаимодействий, *«велика длина описания» (Колмогоров)*;
- **динамичность** – объекты нестабильны и часто изменяют свои структурные и функциональные характеристики;
- **слабая детерминированность** – идентичные элементы объектов могут по-разному реагировать на одинаковое воздействие;
- **открытость** – объекты обмениваются информацией с окружающей средой, в том числе человеком;
- **адаптивность** – объекты обладают коротким временем постоянства,
- **взаимосвязанность** – наблюдается усиление связанности объектов и процессов управления, «природоподобная» глобализация и растущая степень интеграции.

Кроме того, основными свойствами новой цифровой реальности стала возможность жизни и взаимодействия территориально распределенных объектов и процессов в **едином сквозном глобальном информационно-функциональном пространстве с единым глобальным временем.**

В связи с этим безвозвратно устарели архаичные механистические *кибернетические* подходы создания локальных программ, даже если они имеют глобальное использование.

Внедряемые, продвигаемые и анонсируемые решения всех мировых лидеров IBM, ORACLE, SAP, Microsoft, GOOGLE, Яндекс и других, в полной мере служат историческими экспонатами традиционных архаичных подходов. Они скоро канут в лету, если будут и дальше верить в собственные рекламные иллюзии превосходства.

Их борьба с новыми неизбежными трендами может принести лишь недолговременную отсрочку, так как закономерную надвигающуюся концептуальную ИТ-эволюцию не отменить.

ИТАК.

Для цифровой трансформации все объекты и процессы и релевантные им модели и системы управления (инструменты и платформы) необходимо рассматривать как ***«живые (природоподобные) системы»,*** то есть неразрывно целостно и непрерывно в динамике.

Что делать? Где решение?

ЯЗЫКИ ПРОГРАММИРОВАНИЯ. ЭВОЛЮЦИЯ

Большинство «прогрессивных» программистов – *ужасные* ретрограды, они любят повторять *«помните: лучший язык программирования тот, который знаешь в совершенстве»*.

ЯЗЫК ПРОГРАММИРОВАНИЯ (ЯП) – это специальная формальная знаковая система (набор лексических, синтаксических и семантических правил), предназначенная для написания текстов программ для управления компьютером.

ЯП является инструментом программиста и создан для того, чтобы было проще писать и читать тексты команд, но эти специальные тексты затем должны транслироваться (транслятором или интерпретатором) в машинный код, который только и может исполняться компьютером.

За многие годы созданы тысячи различных ЯП от самых примитивных до близких к естественному языку человека. Исторически сложились различные методы систематизации и классификации ЯП.

Чтобы разобраться во всем многообразии ЯП и понимать тенденции развития, выделим следующие основные стадии эволюции языков программирования:

- Машинные коды,
- Языки программирования низкого уровня,
- Языки высокого уровня, Структурное программирование,
- Специализированные языки,
- Параллельное программирование,
- Объектно-ориентированное программирование,
- Визуальное программирование,
- Создание языков сверхвысокого уровня.

МАШИННЫЕ КОДЫ - единственный язык, напрямую понятный ЭВМ. Он реализуется командно-аппаратно, то есть каждую команду выполняет некоторое электронное устройство. Операции являются элементарными, из их последовательностей создаётся вся программа. Ввод программы в цифровом виде производится непосредственно в память с того или иного устройства ЭВМ.

Естественно, что процесс программирования очень трудоемкий, индивидуальный для каждого вида ЭВМ. Читабельность такой программы низкая, разобраться в ней сложно даже автору.

ЯЗЫКИ ПРОГРАММИРОВАНИЯ НИЗКОГО УРОВНЯ (ассемблер, мнемокод, автокод) - мнемонические языки программирования, которые уже имеют определенный синтаксис записи программ. В нём, в частности, цифровой код операции заменен символами (*A – Addition, R – Register* и т.п.). Текст программ получается более читаемый, но их перестаёт понимать ЭВМ.

Поэтому понадобилось создать специальную программу - ***транслятор***, который преобразует программу с языка ассемблера на машинные языки. Практически все ЭВМ имеют свой язык ассемблер. На сегодняшний день ассемблер используется для создания системных программ, использующих специфические аппаратные возможности данного класса ЭВМ.

ЯЗЫКИ ВЫСОКОГО УРОВНЯ, СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ (FORTRAN, PL/1, PASCAL, BASIC, ADA,…) – универсальные алгоритмически полные языки, на которых можно создавать любые прикладные программы. Они имеют широкий спектр типов данных и операций.

Структурное программирование - методология программирования, базирующаяся на системном подходе к анализу, проектированию и реализации программного обеспечения. Эта методология родилась в начале 70-х годов и содержит следующие положения:

- Сложная задача разбивается на более мелкие функциональные задачи. Каждая задача имеет один вход и один выход. В этом случае управляющий поток программы состоит из совокупности элементарных подзадач с ясным функциональным назначением.
- Разработка программы ведётся параллельно и поэтапно. На каждом этапе должно решаться ограниченное число четко поставленных задач с пониманием их значения и роли в контексте всей задачи. Если такое понимание не достигается - значит данный этап слишком велик и его нужно разделить на более элементарные шаги.

Усложняются технологии и система программирования - трансляторы, текстовые редакторы для ввода текстов программ, отладчики для устранения ошибок, специальные программы для создания библиотек программных модулей и множество других служебных программ.

Принципиальными отличиями от языков низкого уровня являются: *использование переменных, возможность записи сложных выражений, расширяемость типов данных за счет конструирования новых типов из базовых, расширяемость набора операций за счет подключения библиотек подпрограмм, слабая зависимость от типа ЭВМ*.

На этих языках создавалось неисчислимое множество различных прикладных программ. Почему же появилось столько языков этого типа, а не один универсальный язык на все случаи жизни?

Просто все разработчики хотят создать самый лучший и самый безошибочный универсальный язык, но оказывается, что проблемы не только в самом языке, они гораздо шире - в особенностях

предметной сферы его использования и в самих принципах разработки программ (основные проблемы жизненного цикла программирования приводятся ниже).

СПЕЦИАЛИЗИРОВАННЫЕ ЯЗЫКИ (Cobol, Snobol, GPSS, Simula, SmallTalk, Modula-2, Ada, Analitic,…) – проблемно-ориентированные языки программирования**,** позволяющие более адекватно описывать объекты и явления реального мира. Они реализуют тенденцию - приблизить язык программирования к языку специалиста в проблемной области.

Например: Cobol - экономические задачи, Snobol - символьная обработка, GPSS, Simula, SmallTalk – моделирование, Modula-2, Ada - задачи реального времени, Analitic - численно-аналитические задачи, и другие.

ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ (для существующих языков программирования разработаны специальные модификации (например, FORTRAN), а также отдельный класс ЯП: Occam, Linda,…) – программирование с распараллеливанием обработки информации в многопроцессорных ЭВМ или распределённых компьютерных сетях с целью ускорения вычислений и эффективного использования ресурсов ЭВМ.

Концептуальная основа параллельного программирования – понятие алгоритма, реализуемого по шагам (событиям) строго последовательно во времени, то есть определяется совокупность параллельно протекающих процессов обработки информации, полностью независимых или связанных между собой пространственно-временными и причинно-следственными отношениями.

Например, язык Occam был создан в 1982 году и предназначен для программирования транспьютеров — многопроцессорных систем распределенной обработки данных. Он описывает взаимодействие параллельных процессов в виде каналов — способов передачи информации от одного процесса к другому.

В 1985 году была предложена модель параллельных вычислений Linda. Основной ее задачей является организация взаимодействия между параллельно выполняющимися процессами.

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ObjectPascal, Java, C++,…) – основа объектно-ориентированного программирования (ООП) заключается в связывании данных с обрабатывающими их процедурами в единое целое - объект. ООП основано на трех важнейших принципах, придающих объектам новые свойства:

- **Инкапсуляция** - объединение в единое целое данных и алгоритмов обработки этих данных. В рамках ООП данные называются полями (параметрами) объекта, а алгоритмы - методами объекта.
- **Наследование** - свойство объектов порождать своих потомков. Объект - потомок автоматически наследует от родителей все поля и методы, потомок дополняется новыми полями и методами или в нём заменяются (перекрываются) методы родителя.
- **Полиморфизм** - свойство родственных объектов решать схожие по смыслу проблемы разными способами.

В настоящее время именно эта парадигма используется в подавляющем большинстве промышленных проектов. Однако нельзя считать, что ООП является наилучшей из методик программирования во всех случаях.

Критические высказывания в адрес ООП указывают на отсутствие значимой разницы в продуктивности разработки программного обеспечения между ООП и процедурным подходом.

Например, многие программисты считают, что наиболее сложной частью создания программного обеспечения является спецификация, дизайн и тестирование концептуальных конструкций, а отнюдь не работа по выражению этих концептуальных конструкций и поэтому ООП не является «серебряной пулей», которая могла бы на порядок снизить сложность разработки программных систем.

Остро критикуются и рекламные материалы об объектном программировании как о некоем всемогущем подходе, который магическим образом устраняет сложность программирования, требует меньше ресурсов или приводит к созданию более качественного ПО.

ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ (Delphi и C++ Builder, C#,..) **-** способ создания программы для ЭВМ с использованием визуальной среды манипулирования графическими объектами интерактивным образом в соответствии с некоторыми правилами вместо написания её

текста (например, редактируются графические объекты и одновременно отображается соответствующий текст программы).

Визуальное программирование часто представляют как следующий этап развития текстовых языков программирования.

В последнее время визуальному программированию стали уделять больше внимания - в связи с развитием мобильных сенсорных устройств, когда использование клавиатуры не очень удобно.

В настоящее время делаются попытки интегрировать подход визуального программирования с программированием потоков данных (dataflow programming).

СОЗДАНИЕ ЯЗЫКОВ СВЕРХВЫСОКОГО УРОВНЯ (Lisp, Haskell, Prolog, к непроцедурным языкам относят языки запросов систем управления базами данных QBE, SQL,..) - непроцедурные ЯП, программист задает отношения между объектами в программе и определяет, что нужно найти, но не задает как получить результат.

Lisp, Prolog создавались как попытки формирования искусственного интеллекта, моделирования мыслительной деятельности человека в процессе поиска решений. Но обещания не соответствовали получаемым результатам и популярность «языков искусственного интеллекта» заметно упала.

Кроме того, программистам сегодня предлагается широко использовать **проблемно-ориентированное проектирование** (DDD - Domain-driven design) - набор принципов и схем, помогающих разработчикам создавать модели объектов и программных абстракций, имитационное моделирование.

Подход DDD рекомендуется в ситуациях, когда разработчик не является специалистом в области разрабатываемого продукта, так как программист не может знать все области, в которых требуется создать ПО.

Однако проблемно-ориентированное проектирование DDD и имитационное моделирование не являются конкретной технологией или методологией получения конечного результата – работоспособного надёжного программного продукта, реальной системы управления.

ИТАК.

Языки программирования развиваются в сторону все большей абстракции от реальных машинных команд. По существу, все современные ЯП сближаются, но вот уже 30-35 лет нет никакого концептуального прорыва.

Что дальше? Каков тренд развития?

ПРОГРАММА. ФОРМА РЕЗУЛЬТАТА

Результат применения программистом языков программирования – ТЕКСТ программы и практически используемый машинный исполняемый КОД.

Программа содержит в том или ином виде **формализованные человеческие знания,** которые являются интеллектуальным продуктом.

Его целесообразно и необходимо накапливать и приумножать.

Однако, как все технические изделия, программы обладают свойствами **естественного физического и морального старения**, одними из причин которых являются и динамика изменения знаний о предметной области, и прямая зависимость от быстро устаревающих ЭВМ, операционных систем, СУБД и другого общего программного обеспечения (ОПО).

При этом формализованные проверенные знания (исторические и актуальные) остаются востребованными, а текст на «древнем» умершем ЯП порой и читать некому.

Как не терять знания? Как их приумножать?

Рассмотрим еще один аспект.

Основные требования, предъявляемые к качеству программного изделия,
- *функциональность, надежность, удобство эксплуатации, безопасность.*

*С*егодня при прочтении, анализе, контролинге, тестировании, сертификации миллионов строк текста **для множества различных модулей программы качество этих характеристик в принципе не установить**.

Даже на одном ЯП тексты множества программ *несопоставимы, семантически не совместимы и не интегрируемы, необозримы, многократно дублируют функциональность в разном семантическом и синтаксическом изложении,*…

А как в тексте выявить логические ошибки и противоречивость функционирования множества модулей сложной программы?

В результате мы имеем *глобальные баснословные финансовые, материальные, технологические, интеллектуальные издержки* на создание миллионов различных прикладных программных продуктов по сути своей реализующих в той или иной степени пересекающиеся, а порой одинаковые задачи предметной области.

Обратите внимание, **ТЕКСТ программы на любом ЯП напрямую совершенно не понятен и совсем не нужен главным участникам-интересантам: *как пользователю, так и вычислительной технике!***

Кроме того, и со стороны специалистов разработчика: *главных конструкторов, архитекторов, постановщиков, аналитиков, экспертов, консультантов, методологов, проектировщиков, тестировщиков, технических писателей и других* – подчас **единственным писателем и читателем текста программы является программист**.

Большие проекты предусматривают совместный труд множества программистов, так они чаще с явным недовольством читают тексты программ друг друга, при этом обычно предлагают написать свой новый лучший текст и за отдельное вознаграждение!

Мировая тенденция на использование «открытых» технологий на уровне открытых текстов программ только на первый взгляд кажется гуманистически прогрессивной философией развития ИТ.

«Открытый» софт даже при наличии армии энтузиастов не позволяет *эффективно* управлять распределенно и параллельно разработанными несопоставимыми версиями.

Открытость текстов программ, как парадигма коллективной распределенной работы, порождает множество изобретателей *«велосипедов»* и, в конечном счете, снижает производительность мирового совокупного труда ничего не знающих о текущей работе друг друга программистов.

Давайте рассмотрим, как принцип - написание программы в виде ТЕКСТА.

Сегодня надо признать, что **ТЕКСТ** программы на любом ЯП – уже **ЭНЕРГЕТИЧЕСКИ НЕ ОПТИМАЛЕН** как форма результата. Текст не воспринимаем ни вычислительным устройством, ни заказчиком. Текст – лишнее звено, пройденный этап развития.

Коэффициент полезного действия растущей армии программистов неуклонно снижается, **энергия и экономика** информатизации цивилизации уходит в «пар», «гудок», «песок»,..

Что делать? Где решение?

ПРОГРАММИРОВАНИЕ. ЖИЗНЕННЫЙ ЦИКЛ

Сегодня программы содержат миллионы строк исходного кода, которые должны правильно и отказоустойчиво исполняться в изменяющихся условиях.

Процесс разработки программы – это реальное промышленное производство, в котором определяющими факторами, как и в традиционной индустрии, являются:

- производительность труда коллектива программистов,
- время,
- себестоимость,
- качество программной продукции.

То есть, разработка программного обеспечения (как и традиционные инженерные дисциплины) сталкивается с проблемами *качества, сроков создания, стоимости, надёжности, скорости и простоты обучения, безопасности, эксплуатации и сопровождения, интеграции, эволюционного развития и модернизации, последующей утилизации,* то есть с понятием **эффективного управления всем жизненным циклом** (ЖЦ)**.**

СТАНДАРТЫ ЖИЗНЕННОГО ЦИКЛА являются описанием создания и использования информационной системы, отражающим его различные состояния, начиная с момента возникновения необходимости в данном изделии и заканчивая моментом его полного выхода из употребления у всех без исключения пользователей.

Наибольшее распространение получили три следующие модели ЖЦ информационных систем:

Каскадная модель (в 70 – 80-е годы) – предполагает переход на следующий этап жизненного цикла системы после полного окончания работ по предыдущему этапу и характеризуется четким разделением данных и процессов их разработки;

Поэтапная модель с промежуточным контролем (в 80–90-е годы) – итерационная модель разработки систем с циклами обратной связи между этапами. Особенностью такой модели заключается в том, что межэтапные корректировки обеспечивают меньшую трудоемкость по сравнению с каскадной моделью; с другой стороны, время жизни каждого из этапов растягивается на весь период разработки;

Спиральная модель (с 90 годов) – делает упор на начальные этапы ЖЦ ИСУ: анализ требований, проектирование спецификаций, предварительное и детальное проектирование. На этих этапах проверяется и обосновывается реализуемость технических решений путем создания прототипов. Каждый виток спирали соответствует поэтапному созданию фрагмента или версии системы, на нем уточняются цели и характеристики проекта, определяется его качество, планируются работы следующего витка спирали. Таким образом, углубляются и последовательно конкретизируются детали проекта, и в результате выбирается некий обоснованный вариант, который доводится до реализации.

В настоящее время эти традиционные жизненные циклы (ТЖЦ) информационных систем регламентированы сближающимися стандартами ИСО, ГОСТ, НАТО и т.п., описывающими множество идентичных этапов и итераций работы:

- Аналитическое исследование:
- Постановка задачи (ТЗ на ИСУ);
- Анализ требований;
- Познание предметной области.
- Декомпозиция на подсистемы (модули).
- Выработка системных соглашений.
- Анализ предметной области при проектировании отдельных подсистем.
- Уточнение требований (итерация 1).
- Разработка программного обеспечения по подсистемам.
- Уточнение требований (итерация 2).
- Тестирование программного обеспечения и выработка замечаний.
- Уточнение требований (итерация 3).
- Доработка программного обеспечения по замечаниям.
- Уточнение требований (итерация 4).
- Разработка документации на подсистемы.
- Тестирование и сдача в опытную эксплуатацию каждой из подсистем.
- Сборка ИСУ и тестирование работоспособности системы в целом.
- Опытная эксплуатация системы, выработка замечаний.
- Уточнение требований (итерация 5).
- Доработка ПО ИСУ в процессе опытной эксплуатации системы.
- Разработка документации на систему.
- Промышленная эксплуатация системы, доработки в процессе промышленной эксплуатации.
- Изменение в реальной предметной области, приводящее к потере адекватности и устареванию (краху) программы.
- Уточнение требований (итерация 6).
- Переход к пункту один и, как правило, смена среды разработки, изменение команды исполнителей.

Во всех стандартах регламентировано, что реализованные этапы, начиная с самых ранних стадий разработки, циклически итерационно многократно повторяются в соответствии с уточнением и познанием предметной области, изменениями требований и внешних условий, введением ограничений, системных соглашений и т.п.

Увеличить >>>

То есть, в целях получения готовой комплексной системы многочисленные коллективы: *менеджеры, главные конструкторы, аналитики, архитекторы, постановщики, разработчики, эксперты, консультанты, программисты, системные администраторы, тестировщики, технические писатели, пользователи и многие другие* **многократно заново проходят десятки шагов традиционного жизненного цикла.**

На каждом этапе ТЖЦ ИСУ порождается определенный набор документов и технических решений, являющихся исходными для последующих этапов. Каждый этап завершается верификацией порожденных документов и решений с целью проверки их соответствия изначально поставленным целям и задачам.

Эти проблемы **кратно увеличиваются и становятся неразрешимыми** при увеличении сложности системы и росте динамики изменения требований.

В итоге, полученное программное обеспечение, реализованное на основе традиционных стандартов жизненного цикла (через месяц, полгода, год,..) – **заведомо ВСЕГДА НЕ АДЕКВАТНО текущей динамике потребности заказчика.**

Информационная система в принципе теряет изоморфность объекту и процессам управления.

Итак, **основные проблемы разработки ПО**:

- **Многоэтапный итерационный жизненный цикл** информационных систем. При создании сложных программ (в соответствии со стандартами) существенно и многократно меняются требования реальной предметной области (у потребителей постоянно возникают новые идеи относительно разрабатываемого программного обеспечения). Информационная система теряет адекватность требованиям и реальной предметной области.
- Недостатки **транспарентности** (прозрачности) процесса разработки:
    - *наборов необозримых текстов программ («чёрных ящиков»),*
    - *оценки персонифицированного вклада,*
    - *стратегического и оперативного планирования (сколько времени займёт разработка),*
    - *состояния проекта (каков процент его завершения),*
    - *структуры финансирования проекта.*
- Недостатки реализации «ручного» **организационно-командного мониторинга и контроля** разработки ПО, которые не позволяют контролировать ход программирования в реальном времени.

- Практически всегда **срываются графики** выполнения работ и **превышаются** установленные **бюджеты**.
- Проблемы **выбора методологии** разработки программного обеспечения. Процесс выбора необходимой методологии может отрицательно отразиться на всех показателях программного обеспечения - гибкости, стоимости, сроках, функциональности.
- **Множество фрагментарных инструментов для каждого этапа** разработки, сопровождения и развития ПО.

Их стоимость, стоимость обучения специалистов и менеджмента владению инструментальными средствами сравнима со стоимостью разработки самой программы.

- **Использование множества моделей объекта на различных этапах проекта**(модель акторов, даталогические модели, инфологические модели, модели структур данных, модель бизнес процессов, ролевая модель, организационная модель, модель событий,..).

Данный подход в дальнейшем кратно увеличивает сложность и запутанность кода. В идеале, при проектировании хочется иметь одну-единственную модель, которая полностью описывает всю предметную область, но в реальности этого до сих ещё никто не реализовал.

- Над сложным проектом и, следовательно, программой работает **большое количество людей** и есть тенденции и практика дробить модель решаемой задачи на несколько более мелких фрагментов. Чем больше людей, тем более значительна данная проблема. В итоге всегда **теряется целостность программы и проекта**.
- **Проблемы интеграции и сборки** отдельных фрагментов программы в единый продукт. Работая над несколькими отдельными модулями и соответственно программами, различные члены проекта могут не знать о сущностях других модулей и программ, что усложняет процесс общей сборки конечного продукта.
- **Недостаточная надёжность**. Постоянное объединение кусков кода от различных разработчиков и проверка работоспособности посредством тестирования.

Самый сложный процесс — поиск и исправление ошибок в программах. А так как число ошибок в программах заранее неизвестно, то неизвестна и продолжительность их отладки.

- **Фрагментарность, противоречивость, нецелостность, отсутствие синхронизации, избыточность, неполнота, неуникальность, несопоставимость** и т.п. данных и методов различных модулей сложной программы;
- **Неадекватность темпов изменения** программного обеспечения реальной динамике изменения требований предметной области;
- Неэффективная интеграция и **нет единого адаптивного информационно-функционального пространства;**
- **Высокие ресурсные издержки** на создание, сопровождение, эксплуатацию, модернизацию и развитие сложных программных систем.
- **Человеческий фактор**, проблема выбора исполнителя.
- **Технологическая, юридическая и финансовая зависимость** от исполнителя.
- И многие другие.

В поисках выхода и в целях повышения эффективности реализации жизненного цикла прикладных информационных систем (постановка, моделирование, проектирование, программирование, исполнение, интеграция, тестирование и т.п.) мировые лидеры IBM, ORACLE, SAP, Microsoft, THALLES и другие, прямолинейно и без лишних раздумий, скупают как бы «лучшие» программные продукты и инструменты для технологической поддержки каждого этапа традиционного ЖЦ.

Они *ищут, выбирают, поглощают, сливают, пытаются объединить и ребрендить* как бы лучшие, но уже концептуально УСТАРЕВШИЕ вчерашние разрозненные технологии.

Даже многократное повышение эффективности каждого из традиционных этапов цепочки традиционного ЖЦ сохраняет *несопоставимость темпов* изменения требований реальной жизни и их реализации в информационных системах.

ИТАК.

Многоэтапные стандарты традиционного жизненного цикла информационных систем **принципиально не могут решить** поставленных проблем и реализовать новые требования.

Их необходимо **полностью концептуально менять**!

Сегодня основные требования к разработке программ включают:

- Принципиальное сокращение цикла разработки программ.
- Увеличение продуктивности работы программистов.
- Улучшение потребительских качеств создаваемых программ.
- Обеспечение адекватности программ динамическим изменениям требований объекта управления.
- Обеспечение продолжительной жизни программы, непрерывную трансформацию «на лету», адаптивность, эволюционность развития.
- Создание, извлечение и применение знаний (накопленного опыта) в режиме реального времени.
- Организация в реальном времени единого информационно-функционального пространства коллективной распределённой разработки.
- Исключение процесса механистической сборки программы.
- Исключение затрат на интеграцию разнородных программ.
- Способность вести большие проекты и группы проектов, обеспечение многоцелевого сбалансированного использования объединенных ресурсов и процессов.
- Возможность повторного мультицелевого использования уже созданного ПО.
- Формирование единого экспертного пространства;
- Повышение надежности и безопасности сложных систем управления;
- Многократное уменьшение объемов ресурсов и затрат на разработку, сопровождение, эксплуатацию и развитие информационных систем.

Существующие традиционные подходы SAP, Oracle, Microsoft, IBM, THALES, GOOGLE и других к разработке информационных систем **не соответствуют** этим требованиям.

ПРОГРАММИСТ. ЧЕЛОВЕЧЕСКИЙ ФАКТОР

Традиционный подход к созданию программ чётко разделяет людей на две категории:

- **Заказчиков:** постановщиков задачи, специалистов в той или иной предметной области, методологов, пользователей системы и других.
- **Исполнителей:** менеджеров**,** разработчиков, главных конструкторов, архитекторов, экспертов, проектировщиков, аналитиков, консультантов, методологов, программистов, тестировщиков, технических писателей и других.

Одни являются экспертами в предметной области, другие обладают монополией на знания в области IT-технологий. Взаимодействие между двумя этими группами людей, как правило, трудно согласовывается, что оказывает влияние на качество разрабатываемых систем.

Сформировалось устойчивое профессиональное сообщество, которому очень на руку такое положение вещей.

Это – ПРОГРАММИСТЫ.

Они выступают в роли неких *«жрецов»*, постепенно **становясь не только проводником, но и барьером между обществом и информацией.**

Приведем ряд проблем, с которыми сталкиваются возможности разработчика и ожидания заказчика:

- разработчик не всегда располагает исчерпывающей информацией для оценки требований к информационной системе с точки зрения заказчика;
- заказчик, в свою очередь, не имеет достаточной информации о проблемах обработки данных для того, чтобы судить, что выполнимо, а что нет;
- разработчик сталкивается с чрезмерным количеством не систематизированных подробных сведений, как о предметной области, так и о новой информационной системе;

- традиционная текстовая или графическая спецификация информационной системы из-за её объема и использования технических терминов часто не обозрима и не понятна заказчику;
- если спецификация текстовая и графическая понятна заказчику, то она порой недостаточна для проектировщиков и программистов, создающих или адаптирующих информационную систему;
- любая традиционная текстовая и графическая спецификация информационной системы – это обсуждение заказчиком и разработчиком уже «вчерашних» требований к этой системе.

Описание любой задачи, более или менее согласованное между заказчиком и разработчиком, претерпевает дополнительные многократные искажения при прочтении и воплощении его многочисленной армией главных конструкторов, архитекторов, проектировщиков, программистов и других.

И чем дальше от заказчика – тем больше искажений. Важно, наконец, сократить расстояние между специалистом и необходимым ему программной системой, убрать многочисленных толмачей «с русского на русский». Игру в *«испорченный телефон»* необходимо оставить для досуга.

Конечно, применение методик системной и программной инженерии, ключевое место среди которых занимают методологии структурного и объектно-ориентированного анализа, снимает отдельные проблемы, однако не даёт решение совокупности этих проблем.

Традиционные подходы к созданию прикладных программных продуктов приводят к тому, что разработчик или группа разработчиков начинают свою деятельность практически каждый раз с нуля, то есть с разрозненного личного опыта, накопленных заготовок (порой устаревших), наличия информации об унаследованных аналогах. Нет эффективных инструментов обобщения накопленных знаний и опыта миллионов программистов, а также их эффективного использования.

Сегодня качество создаваемых программных продуктов зависит, прежде всего, от *профессионализма, культуры, компетентности, таланта, опыта, привычек, психологических личностных характеристик конкретных специалистов*.

Распространена экстенсивная стратегия увеличения числа программистов, занятых в разработке информационных систем, развития оффшорных зон программирования (Индия, Китай, Ирландия, Израиль, Россия и др.).

Но во всём мире растет общее недовольство деятельностью уже столь многочисленной касты программистов:

дорого, долго, некачественно, результаты не адекватны требуемой задаче, формируется зависимость от конкретных исполнителей, проблема внести даже самые малые изменения в

«черные ящики» унаследованных программ, в программах никто не хочет разбираться, и как следствие - бесконечное обоснование предложений начать все сначала и опять писать миллионы строк текстов программ с непрогнозируемым завершением работ…

Процесс – всё, результат – ничто.

Пришло осознание, что необходимы новые методы «безлюдного» глобального производства программного продукта. Одним из основных индикаторов построения информационного общества послужит резкое сокращение потребности в программистах, которых заменят инновационные технологии создания прикладных программных систем.

КИБЕРБЕЗОПАСНОСТЬ

В целях обеспечения кибербезопасности можно «закрыть» физический контур информационной системы, поставить везде видеонаблюдение, бороться с вирусами, отключиться от интернета, не пускать к системе с современными гаджетами, проверять персонал на «детекторе лжи», шифровать данные в каналах и на серверах и т.п.

Однако все эти усилия могут быть тщетны, так как современная сложная информационная система состоит из множества модулей различных функциональных программ, которые имеют несовместимые индивидуальные технологические интерпретации обеспечения того или иного уровня кибербезопасности, в том числе ролевого и мандатного доступа (защиты от несанкционированного доступа к ПО).

В тексте любой, даже небольшой **функциональной** программы сложнее всего найти **недекларированные (недокументированные) возможности** (НДВ, undocumented features) — это возможности программного обеспечения, не отраженные в документации. Частным случаем недокументированных возможностей являются недокументированные функции.

Часто недокументированные возможности сознательно закладываются разработчиками в целях тестирования, дальнейшего расширения функциональности, обеспечения совместимости или же в целях скрытого контроля за пользователем.

Преднамеренно внесённые в ПО функциональные объекты (уязвимости), обладающие недекларированными возможностями, - **программные закладки**.

Недокументированные возможности могут стать так же следствием не учтенных разработчиками побочных эффектов и ошибок (баги, дыры,..).

Недокументированные возможности обнаруживаются, обычно, в процессе обратной разработки («reverse engineering»), но могут быть обнаружены и случайно.

При любой сертификации ПО **комплексная безопасность** множества фрагментарных прикладных информационных систем в принципе НЕ ДОСТИЖИМА. В целом она всегда будет определяться программой с наименьшим уровнем безопасности - **самым «слабым звеном»**.

По факту для специалиста (в том числе хакера) всё комплексное программное обеспечение представляет собой *«решето»* со всеми возможными уязвимостями, в том числе утечки информации и не санкционированного доступа.

Сегодня нет эффективных технологических средств поддержки разработчиков в единой согласованной, но динамически развиваемой в ходе реализации проекта, целостной концепции безопасности.

К тому же сложилась общемировая тенденция, когда множество программных продуктов прославленных брендов (в том числе и в интересах силовых министерств) производятся по факту на «глубоком» аутсорсинге с большим количеством национальных и международных посредников и в конце концов (ради экономии) студентами старших курсов «на коленках».

ИТАК.

В настоящее время сертификация сложных функциональных программных систем и комплексов на недекларированные (незаявленные) возможности – длительная дорогостоящая процедура с сомнительным результатом.

Разбор тысяч и миллионов строк текста функциональных программ на скрытые «закладки» (например, когда центрифуга может раскрутиться до саморазрушения, а ракета уничтожить собственные позиции) по существу НЕ РЕАЛИЗУЕМ.

В мире ни один уполномоченный орган, сертифицирующий программное обеспечение не может вам гарантировать, что *«мин нет»*!

Все, что он может вам гарантировать – это то, что **«недекларированные возможности не обнаружены»**.

Каждая сертификация комплексного программного обеспечения требует много времени и других ресурсов.

А **как сертифицировать «живые» программные системы с высокой динамикой изменений?**

РОБОТ ПО ПРОГРАММИРОВАНИЮ

«Умных» гаджетов с каждым годом становится всё больше. За красивой объединяющей метафорой *«интернет вещей» (Internet of Things, IoT)* или *«интернет всего» (Internet of Everything)* сегодня примитивно скрываются миллионы «лоскутных» автономных программ.

Apple, Google, Microsoft и другие решили, что *«для масштабной работы различных систем необходим единый язык».* Они все сейчас работают над своими *«экосистемами»,* но по отдельности, в разные стороны, а значит, в лучшем случае все получат опять локальные системы, которые опять сложно будет объединить.

НОВАЯ ПАРАДИГМА.

Итак, оценив ключевые аспекты текущего состояния эволюции программирования, можем сделать следующие заключения:

- **ОБЪЕКТЫ УПРАВЛЕНИЯ**: сложным взаимосвязанным живым объектам и процессам должны соответствовать «живые» сложные природоподобные информационные системы управления.
- **ЯЗЫКИ ПРОГРАММИРОВАНИЯ**: все языки программирования не понятны и не нужны ни заказчику, ни вычислительной технике. Они катастрофически теряют эффективность при коллективной распределенной работе множества программистов.
- **ПРОГРАММА:** текст программы как форма результата работы программистов для создания «живых» информационных систем - энергетически и экономически не оптимален.
- **ПРОГРАММИРОВАНИЕ**: традиционный итерационный многоэтапный жизненный цикл архаичен и в принципе не может соответствовать динамике изменения требований сложных динамических систем.
- **ЧЕЛОВЕЧЕСКИЙ ФАКТОР:** программисты как посредники тормозят рост производительности в создании сложных динамических «живых» информационных систем.
- **КИБЕРБЕЗОПАСНОСТЬ**: традиционные подходы в принципе не могут обеспечить безопасности сложных динамических «живых» информационных систем.

Вывод один – необходимы инновационные конструктивно оригинальные идеи выхода из традиционного направления затухающего развития, в котором близок и обозрим тупик.

Мы не прогнозируем будущее, мы его создаём.

В то время как все (кто во что горазд) из отдельных модулей строят информационное общество, мы, исследуя этапы социально-экономического развития: *доаграрный, аграрный, индустриальный, информационный,* **задались вопросом:**

что надо придумать и сделать, чтобы успешно завершить строительство информационной эпохи, приблизить КОНЕЦ информационного общества и пойти дальше?

Анализ некоторых аспектов завершения предыдущих эпох показал:

- **аграрная эпоха** привела к повышению эффективности крестьянского труда за счёт механизации и изъятия человека из трудоемких процессов, 15-20% населения планеты сегодня способны накормить весь мир,
- **индустриальная эпоха** закончилась «уничтожением» пролетариата – «движущей силы революции». Лозунг «пролетарии всех стран объединяйтесь» потерял свою актуальность. Сегодня 15-20% населения могут нас обеспечить всеми необходимыми производственными товарами за счет новых технологий, роботизации и автоматизации.

Было сделано предположение: а что, если вообще НЕ ПИСАТЬ тексты программ для создания сложных динамических «живых» функциональных информационных систем, принципиально устранить «самые слабые звенья»:

Создать «робот», «станок» по программированию и максимально изъять из процесса «строительства» информационного общества основную движущую силу – программистов.

Как этот робот должен работать?

Природоподобно!

Для поиска новых подходов создания сложных искусственных систем (производимых человеком), в том числе информационных систем, пора посмотреть на жизнь биологических природных систем.

До настоящего времени рядом с природой человек все искусственные системы создавал креационно из частей-модулей механистично (сборка/разборка/замена). Научно-технический прогресс, «эволюция» и «развитие», всех антропогенных систем (даже в биологии, медицине, климатике, социологии,..) пока идут на тех же редукционных принципах (включая эко, киборгов, гмо, селекцию, трансплантацию, экзоскелеты, нанороботов, искусственный интеллект,..).

Если природные биологические системы сами воспроизводят новые (себе подобные) системы и эволюционируют в новые виды, то невозможно представить, например, атомную электростанцию, которая кроме энергии ещё и строила бы другие электростанции и «бесшовно» эволюционировала в термоядерную энергетику.

Каковы принципы эволюционной динамики живого?

Где и как живет «программа», которая определяет возможности сложной и динамической жизнедеятельности, роста, развития и размножения всех современных организмов, их самоорганизацию?

Природа умудрилась в одной длинной полимерной макромолекуле ДНК (единой записи в виде двудольного гиперграфа, содержащего всего-то около трех миллиардов генов), изначально целостно описать всю *«программу»*[1] **автоматического** преобразования этого «упакованного» описания в реальные объекты и процессы всего живого от зарождения до смерти.

&&&

[1] *Ген* — единица передачи наследственной информации и участок ДНК, который влияет на определённую характеристику организма.

Генотип — совокупность всех генов организма, являющихся его наследственной основой.

Фенотип — совокупность внешних и внутренних признаков организма, приобретённых в результате онтогенеза (индивидуального развития).

&&&

Так, геном человека изначально содержит *модель знаний:* какие будут у него руки, ноги, нос, глаза, склонность к болезням, черты характера, как он будет расти, как он будет стареть,..

Природа как невидимый робот **автоматически «материализует» модель знаний человеческого ДНК из клетки в реальную физическую сложную динамическую систему *Homo Sapiens*.** Фенотипически мы только индивидуально используем, правим, усиливаем или разрушаем заложенное в нас от рождения.

По сути ДНК можно рассматривать как один длинный целостный информационный файл – материальную модель описаний, которая представляет собой граф - набор взаимосвязанных вершин.

При этом количество *видов* вершин и *типов* связей в этом графе конечно и не велико. Так, например, в ДНК есть только четыре вида азотистых оснований *аденин, гуанин, тимин, цитозин* и они строго ограничены по возможным комплементарным связям.

То есть неправ фон Нейман и его последователи, утверждающие, что *простейшим описанием объекта, достигшего некоторого порога сложности, оказывается сам объект, а любая попытка его строгого формального описания приводит к чему-то более трудному и запутанному*.

У природы отлично получается **упаковывать «знания» в компактные «информационные» материальные модели адекватные сложному вещественному динамично развивающемуся объекту**, следовательно, выход есть.

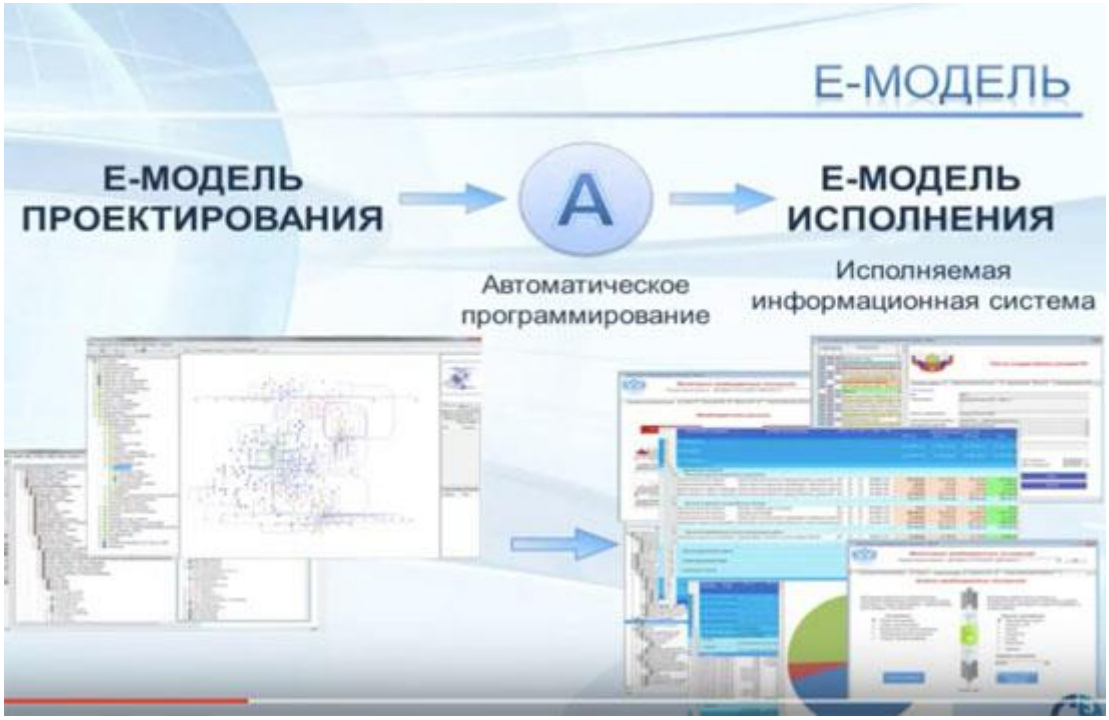
В России создана теория, методология и совокупность **GGG-**технологий **(G3, Graph, Global Gnoseology Graph, Глобальный Гносеологический Граф, гиперграф Хохловой)** новой поствинеровской кибернетики.

Впервые сложные искусственные информационные системы создаются не механистически (методом антропогенной сборки), а «выращиваются» на основе природных «биологических» принципов, включающих «наследование».

GGG-подход включает следующие основные технологии»:

- **G3A** – сетецентрическая архитектура систем (net-centric architecture);
- **G3LC** – «биологический» жизненный цикл систем (software «bio» life cycle);
- **G3L** – визуальный язык моделирования знаний (visual modelling language);

- **G3EM** – коллективное эволюционное моделирование систем (software evolutionary modelling);
- **G3AP** – автоматическое программирование систем (automatic programming);
- **G3I** – интеграция, семантическая интероперабельность систем (integrator);
- **GGG – GRAPH** – новая глобальная сеть, (intellectnet);
- **G3S** – глобальная модель и база знаний (global knowledge model&database);
- **G3SEC** – глобальная информационная безопасность (global cybersecurity);
- **G3WG** – глобальное сетецентрическое управление (global net-centric management).
- И другие.



Предлагается больше не тратить время и усилия на совершенствование языков программирования, а пойти с другой стороны - создать **единый общий универсальный эволюционно развивающийся язык описания человеческих знаний о любой предметной области.**

Сегодня наряду с многонациональными естественными языками разнодисциплинарные знания уже имеют свои локальные, но единые глобально унифицированные семантические и символические языки описания *математики, музыки, физики, медицины, химических реакций, неорганической химии, экономики, бухгалтерского учета, управления дорожным движением, навигации на олимпиадах, эмоциональной реакции на информацию, картографии,..*

Но Мир - един и все эти языки фрагментарно описывают одни и те же предметы и процессы реальной жизни, только с разных дисциплинарных фокусов, поэтому все локальные языки описания знаний с одной стороны произвольно асинхронно пересекаются, а с другой - несопоставимы.

Увеличить >>>

Выход из тупика надо искать реализуя переход на новый, более высокий уровень абстракции, выделив общие закономерности семантического и символического описания различных знаний об одном целостном предмете и процессе.

Было время разбрасывать камни, приходит время собирать.

Предложен новый метод и единый глобальный универсальный язык коллективного кросс-дисциплинарного описания Единой Модели Знаний цивилизации для создания и развития **единой целостной картины мира.**

При реализации новых природоподобных подходов создания человеком искусственных программных систем опровергнуто ключевое утверждение, что *простейшим описанием объекта, достигшего некоторого порога сложности, является сам объект.*

На основе **GGG**-технологий коллективно распределенно формируется единый целостный эволюционно развиваемый один файл (как *«цифровое ДНК»*) – **информационная модель знаний об объектах и процессах** (в виде сетецентрического гиперграфа Хохловой), на основе которого **виртуальным роботом автоматически**, без участия программистов, без формирования и редактирования текстов программ, сразу создаётся готовая **исполняемая информационная система управления** релевантная предметной области, описанной в модели.

GGG-технологии производят продукты прямого пользования в интересах только 2-х участников:

- **Пользователю (заказчику)** - виртуальная читабельная модель знаний и функционирующая информационная система управления теми или иными объектами и процессами, в том числе с возможностью имитационного моделирования.
- **Вычислительной технике** - исполняемый код.

При этом принципиально изменяется архитектура функционального программного обеспечения - информационной системы управления.

Предлагаются новые подходы к реализации дальнейшей эволюции архитектуры ПО:

монолитная – модульная – сервис-ориентированная – СЕТЕЦЕНТРИЧЕСКАЯ

Где:

- **Монолитная архитектура:** одна функциональная задача - одна информационная система**.**
- **Модульная архитектура:** одна комплексная функциональная задача - интеграция множества модулей информационных подсистем методами каждый с каждым, экспорт/импорт, P2P,…

- **Сервис-ориентированная архитектура:** одна комплексная функциональная задача - множество модулей информационных функциональных подсистем «интегрируется» множеством модулей информационных подсистем интеграции.
- **СЕТЕЦЕНТРИЧЕСКАЯ архитектура**: множество комплексных функциональных динамических задач - единая сетевая мульти-целевая информационно-функциональная среда.

Множество итерационных этапов традиционного стандарта жизненного цикла информационных систем сведено к предельно возможному минимуму ресурсных затрат.

Инновационный жизненный цикл информационных систем **G3LC** содержит только **ДВЕ СТАДИИ** с автоматическим переходом между ними:

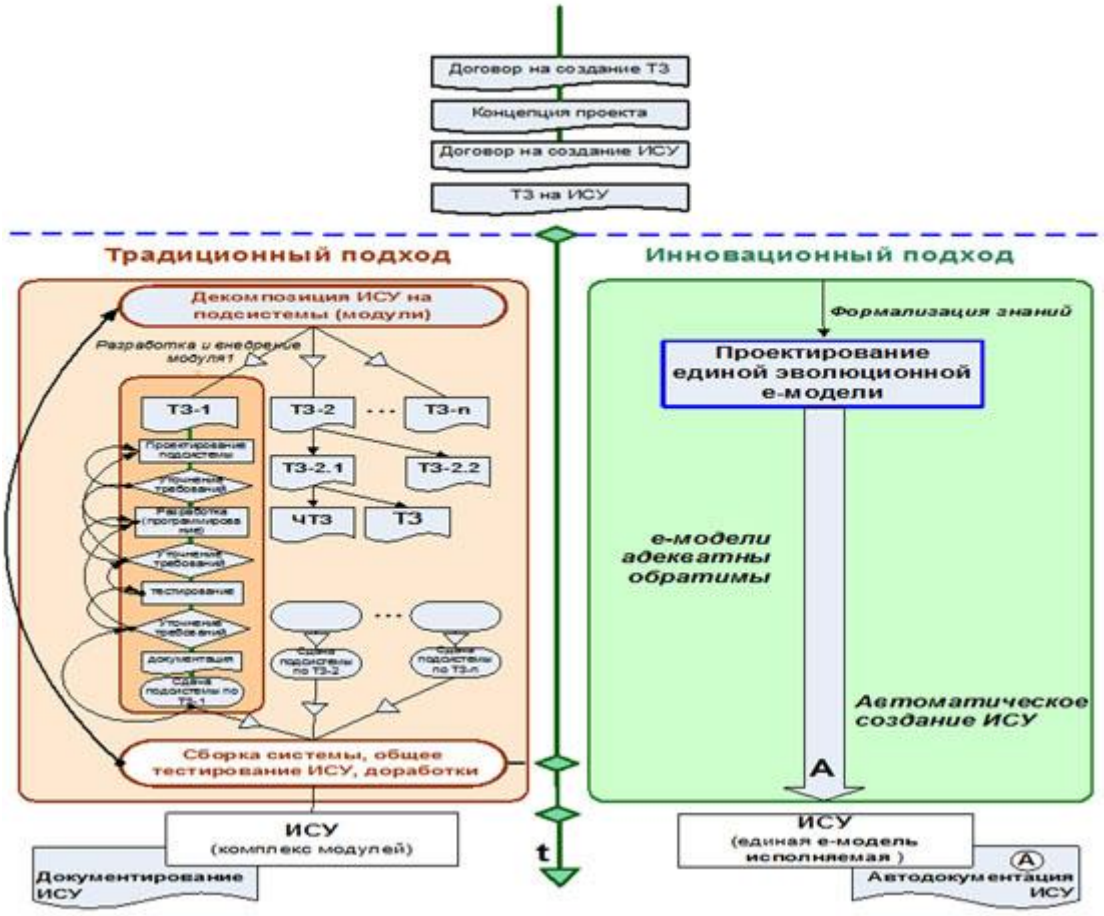
- **проектирование** *(planning)* сетецентрической **G3S-**модели знаний об объекте*;*
- **исполнение** *(running)* функциональной сетецентрической информационной **G3-**системы*.*

И всё.

Система всегда находится только в двух обратимых и адекватных друг другу состояниях: описания и исполнения.

Особо обращаем внимание, что в **GGG**-парадигме концептуально и осознанно отказались от промежуточного этапа автоматического формирования текстов программ, их редактирования, оптимизации, дописывания и отладки, как это пытаются делать наши современники (ARIS, UML и все другие средства генерации текста программы по моделям).

Дело в том, что когда программистами вносятся малейшие изменения в автоматически сформированные строчки текста программы, то последняя теряет изоморфность своим моделям.



Увеличить >>>

Специалист обучается только двум навыкам – эволюционно моделировать свои знания о предметной области и оперативно пользоваться готовой информационной системой.

Концептуально сокращено количество «посредников» между заказчиком и результатом. В предлагаемой **GGG**-методологии реализована следующая специализация:

- **системный аналитик** (эксперт, методолог, исследователь, пользователь,..) – проектирование модели знаний о предметной области;
- **пользователь** – эксплуатация функциональной информационной системы.

Системный аналитик и пользователь могут объединиться в одном человеке.

Создание **робота по программированию** возможно только при соблюдении всей совокупности необходимых и достаточных принципов и условий, остановимся на пяти ключевых.

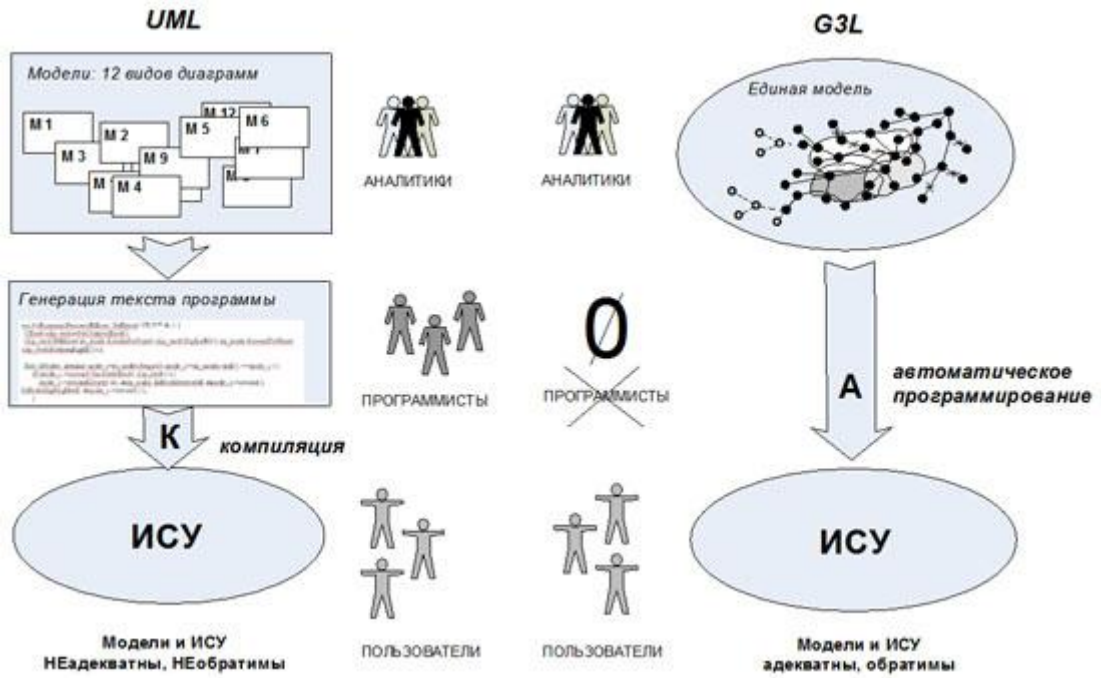
1.**ЕДИНАЯ И ЕДИНСТВЕННАЯ G3S-модель знаний** формируется в одном информационном файле, содержащем полное и непротиворечивое описание коллективных знаний об объектах и процессах предметной области *(содержание, форма, поведение)* так, что далее оно автоматически (роботом, без участия человека) преобразуется в понятные действия компьютера *(памяти, процессора, интерфейсного устройства).*

Поскольку в новой технологии все информационные системы эволюционно «выращиваются» и автоматически создаются из единой целостной G3S-модели: «информационной ДНК», то МЕЖМОДУЛЬНЫХ ИНТЕРФЕЙСОВ НЕТ по определению, а, значит, нет и сборки программ в единый проект.

Все наши оппоненты

создают множество моделей описания различных аспектов предметной области *(модели акторов, событий, классов, объектов, пакетов, деятельности, инфо-логические, дата-*

логические, имитационные, компонентов, композитной/составной структуры, баз данных, развёртывания, вариантов использования, коммуникации и последовательности, синхронизации,...) – и поэтому **никогда не смогут** полностью автоматизировать программирование.



Увеличить >>>

2.В G3S-модели знаний **КОЛИЧЕСТВО ВИДОВ ЭЛЕМЕНТОВ** (классов и связей) (С описанием видов классов и типов связей G3S-модели знаний можно ознакомиться а работах «Теория эволюционного моделирования», «О теории моделирования и гиперграфе классов» и других.), на основе которых эволюционно (наследованием классов) описывается гиперграф модели знаний о любой предметной области, – **КОНЕЧНО И НЕ ВЕЛИКО** (количество классов на фундаментальном уровне наследования – 3, бинарных связей – 7, множественных связей – 3). При этом модель знания о любой предметной области «выращивается» и сочетается из этих элементов (по аналогии с конечным числом химических элементов таблицы Менделеева, из которых создан весь мир).

В противном случае задача создания робота по программированию является неразрешимой.

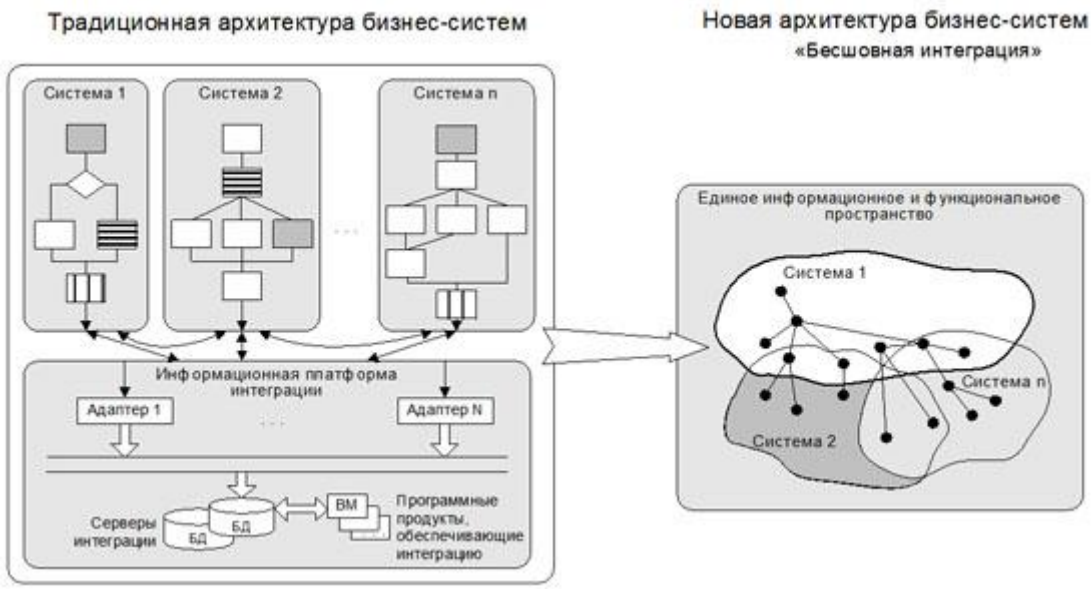
Все наши оппоненты

описывают знания о предметной области с помощью субъективных сочетаний и перестановок слов естественного языка в виде семантических сетей, онтологий, экспертных систем, имитационных моделей, ссылочных структур, OWL-подобных примитивных триплетных конструкций (*«мама мыла раму», «Маша любит кашу»)* и т.п..

Где все классы - вершины (чаще существительные) и связи (чаще глаголы) формируют **бесконечное множество** субъективных элементов (слов и символов) описаний предметной области.

На основе бесконечного множества элементов описаний этих «знаний» о какой-либо предметной области НИКОГДА НЕ СОЗДАТЬ робота автоматического программирования информационных систем управления данной предметной областью, то есть эти «знания» невозможно системно использовать и на практике.

Следовательно, невозможно проверить достоверность этих описаний – истинность и ложность этих как бы «знаний».



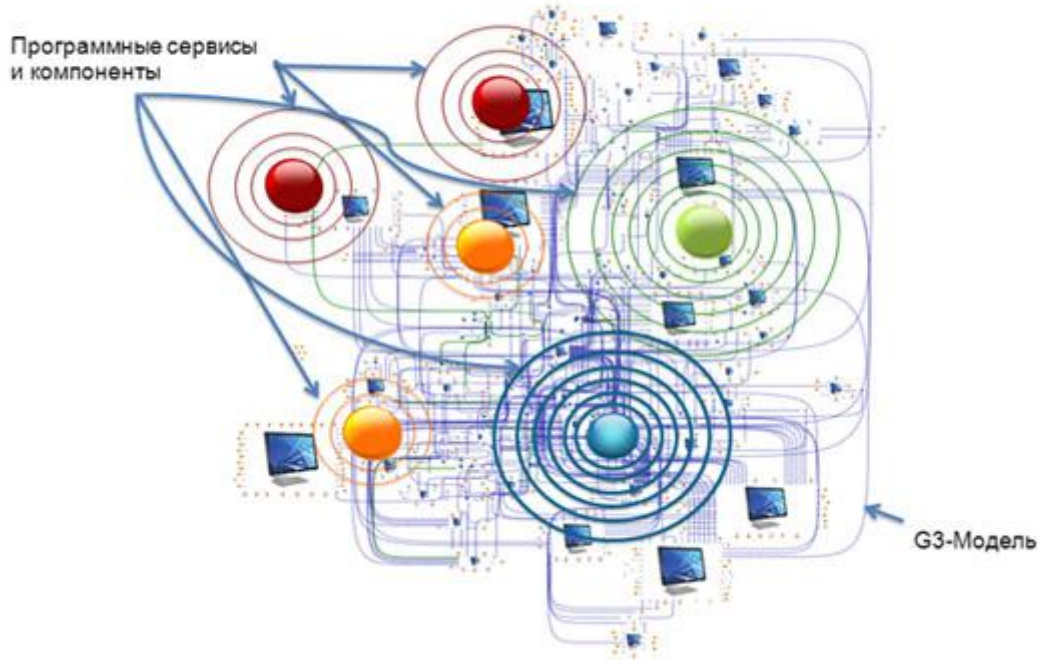
Увеличить >>>

3.G3S-модель знаний - **МУЛЬТИЦЕЛЕВОЙ ОДНОРАНГОВЫЙ СЕТЕЦЕНТРИЧЕСКИЙ ГИПЕРГРАФ** классов (гиперграф Хохловой). Знания *о вселенной, атоме, клетке, экономике, человеке, климате, самолёте, медицине и т.п.* лежат в одной целостный сквозной одноранговой сети.

Они естественно «бесшовно» взаимодействуют и не имеют «преимущества» друг перед другом.

В одноранговой мультицелевой сети этого G3-гиперграфа могут быть сформированы динамические произвольные иерархии приоритетов целевого рассмотрения или решения конкретных практических задач управления.

То есть можно как бы потянуть за какую-либо вершину - предметно-ориентированный класс гиперграфа и «увидеть» степень и иерархию уровней взаимосвязанности и взаимовлияния данного предмета относительно других.



В G3S-модели нет понятия внешней среды, она – часть общей модели. Визуальная графовая модель открыта и читабельна с помощью изменяемого множества сценариев контейнеризации, фокусирования, навигации, фильтрации,.. классов и связей гиперграфа Хохловой.

G3S – эволюционно коллективно и распределенно создаваемая *единая целостная модель кросс-дисциплинарных знаний цивилизации для мультицелевого использования.*

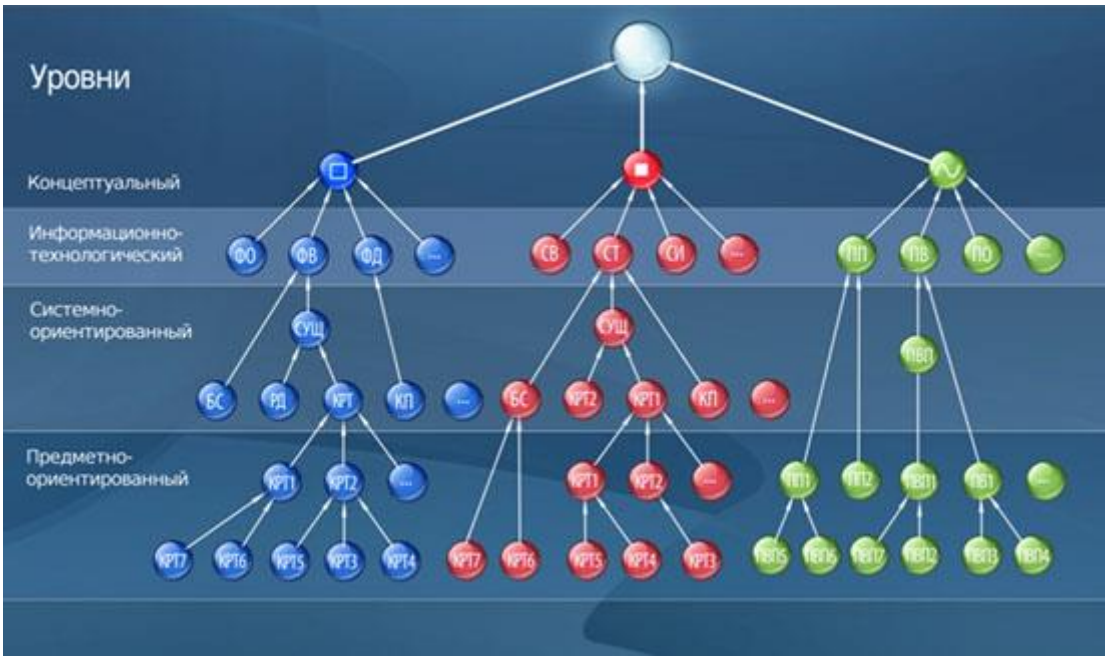
Все наши оппоненты

описывают модели знаний тоже, казалось бы, графами, но не теми.

Они примитивно «в лобовую» формируют **фрагментарные** *«модели знаний», семантические сети, онтологии,...* креационно из слов естественного языка (вершины и ребра графа) для локальных и конкретных целей и задач, а также по различным принципам их классификации: *дисциплинарные, функциональные, отраслевые, для различных уровней образования, территориальные, корпоративные и т.п.*

Количество создаваемых семантических вершин и связующих их семантических ребер отдельных графов – *«моделей знаний» различных производителей слабо формализовано и не ограниченно велико (стремится к бесконечности).*

Из этих «обрывков» НИКОГДА не склеить целостную модель знаний цивилизации и не реализовать робота по программированию.

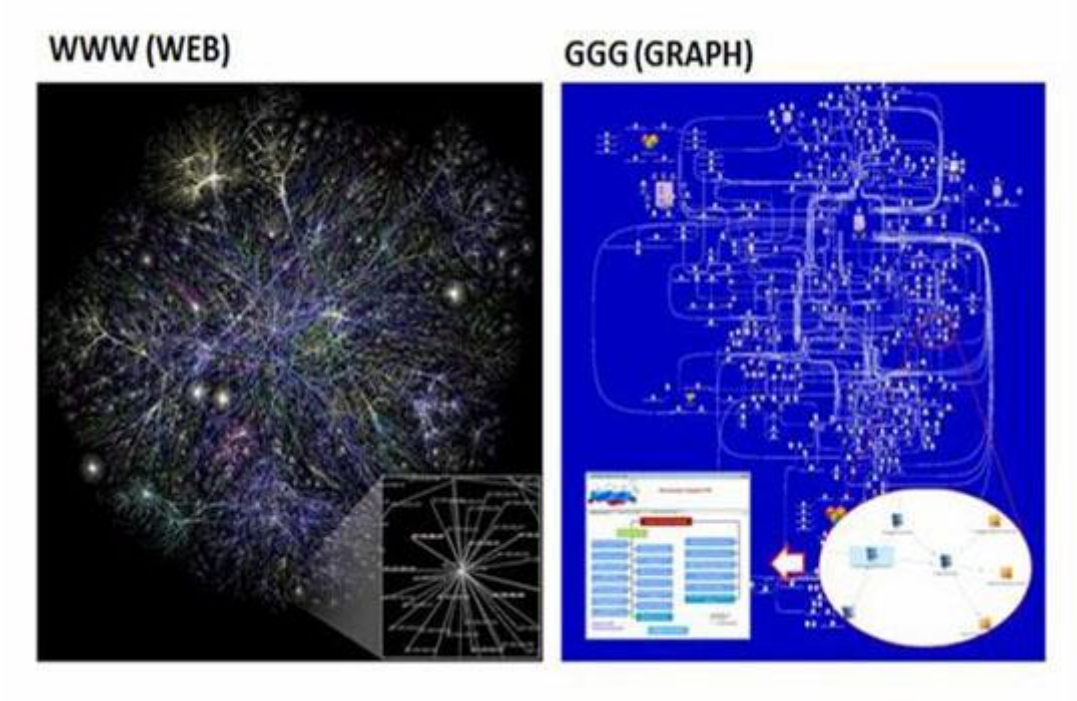


Увеличить >>>

4.G3S-модель знаний создаётся на основе строгой формализации одновременного применения **ЭВОЛЮЦИОННЫХ И КРЕАЦИОННЫХ МЕТОДОВ**.

Все классы (вершины) гиперграфа Хохловой создаются только эволюционно - наследованием, то есть модель знаний G3S развивается целостно наследованием из **одного прокласса**.

Креационно добавляются новые свойства классов (параметры и методы) а также бинарные связи структуризации (3 вида) и синтеза (3 вида) классов.



Наследование классов обеспечивает уникальные свойства G3S-модели знаний: *технологическую автоматическую системность, классифицируемость, читабельность, надёжность, безопасность, использование чужого опыта, формирование и автоматическую поддержку динамически развиваемых стандартов и многие другие.*

Все наши оппоненты

при проектировании моделей знаний создают вершины их свойства и связи креационно, то есть на «чистом» листе, экране,.. «рисуют» (копируют) новые вершины (точки, символы из списка или библиотеки символов) и связывают их логическими семантическими связями.

Термин наследование часто ошибочно применяется к методу объединения понятий по какому-либо множеству признаков, не реализуя ни существо самого метода, ни принципов наследования.

Например, вводится понятие родителя - «собака», от которого как бы наследуются «Жучка», «Бобик», «Альма», различные породы и т.п.

Модели знаний, создаваемые креационно, НИКОГДА не смогут послужить основой для создания и применения робота по программированию.

5.G3S-модель знаний создается **РАСПРЕДЕЛЁННО КОЛЛЕКТИВНО КОНВЕРГЕНТНО в РЕАЛЬНОМ ВРЕМЕНИ.**

В G3-гиперграфе применяются методы САМООРГАНИЗАЦИИ[1] (без единого руководящего центра), взаимоконтроля и «бесшовного» взаимодействия.

Каждый может высказать и описать свою научную гипотезу в общей модели знаний (исторической, актуальной, прогнозной) – G3-информационной «ДНК» и, выполнив автоматическое программирование, получить, исполнить и проверить на практике работоспособность этой гипотезы и ее влияния как на систему в целом, так и на любую произвольно выбранную часть целого.

То есть можно имитационно-прогностически поработать со своеобразной «селезенкой», «сердцем», «пальцем»,.. единого целостного организма.

В GGG-подходе на новом уровне абстракции гармонично реализуются противоречивые стремления человечества к проявлению индивидуальности, независимости, свободе, персонификации авторства, открытости и транспарентности процесса создания информационных систем.

Режимы сборки и интеграции в сетецентрических G3-моделях и системах отсутствуют в принципе.

Все наши оппоненты

До настоящего времени большинство усилий направлено на создание универсальных шлюзов и технологий формирования и сопровождения «среды общения» и «обмена сообщениями» между несопоставимыми программами «черными ящиками».

Ожидается открытие секрета этакого универсального «клея» для воды, камня, огня, газа, бумаги, кислоты и т.п.

Но попытки интеграции прикладных программ на уровне создания универсальных «интерфейсов черных ящиков» консервируют и еще больше обостряют проблему создания единого достоверного информационно-функционального пространства и эффективного адекватного управления.

Увеличить >>>

Можно привести ещё много уникальных свойств G3S-модели знаний, которые более полно и подробно описаны в ряде книг, статей и эксплуатационной документации к GGG-технологиям.

Однако задачами данной главы является первичное знакомство с новой парадигмой создания РОБОТА по ПРОГРАММИРОВАНИЮ **G3AP.**

ЗАКЛЮЧЕНИЕ

Создание РОБОТА ПО ПРОГРАММИРОВАНИЮ – закономерная эволюция разработки программных продуктов, естественное явление технологического развития цивилизации, эволюции информационного общества и «цифровой экономики».

В ближайшем будущем останется потребность только в той доле программистов, которые будут совершенствовать информационные инструменты и инфраструктуры взаимодействия человека и вычислительного средства, в том числе автоматического программирования.

Самый продолжительный этап − разработка ТЕКСТА программы и отладка программного кода − исчезает, а значит, время и стоимость проектирования сложных систем сокращается на порядки.

Предложен новый метод и единый глобальный универсальный язык коллективного кросс-дисциплинарного описания ЕДИНОЙ МОДЕЛИ ЗНАНИЙ цивилизации для создания и развития единой целостной КАРТИНЫ МИРА**.**

2-х этапный ЖИЗНЕННЫЙ ЦИКЛ G3LC позволяет получать работоспособное промышленное программное обеспечение уже на ранних стадиях проектирования модели знаний и наращивать возможности системы эволюционно при текущей продуктивной промышленной эксплуатации.

Обеспечивается СОПОСТАВИМОСТЬ ТЕМПОВ АДАПТАЦИИ информационных систем и темпов изменения требований и задач заказчика, возможность переноса идей и опыта развития из одних предметных областей в другие, возможность сознательного и целенаправленного управления развитием.

Робот по программированию реализуют новое качество НАДЁЖНОСТИ и БЕЗОПАСНОСТИ сложных «живых» информационных систем.

GGG-технологии на этапах моделирования информационных системы обеспечивает возможность участия именно специалистов (экспертов) предметной области, без использования программистов. Исчезает барьер между обществом и информацией.

Программисты (подобно луддитам) уже вступили в борьбу с «е-станками» автоматического программирования G3AP, но и этот слой современного «пролетариата цифровой экономики» подлежит естественному эволюционному сокращению.

Создание и оперативное развитие адаптивных сетецентрических информационных GGG-систем управления позволит в свою очередь на новом качественном уровне оптимизировать ГЛОБАЛЬНОЕ КОЛЛЕКТИВНОЕ научно-обоснованное УПРАВЛЕНИЕ.

Многие управленческие процессы динамически будут GGG-трансформироваться в автоматические сценарные режимы «автопилотирования», принципиально сокращающие бюрократию и «офисный планктон».

ФАКТОРЫ УСПЕХА

Вольное сетевое сообщество на единой технологической G3-платформе объединяет результаты коллективной инновационной деятельности специалистов-участников более 1000 организаций (бизнес и государственных структур, научных институтов, образовательных и медицинских учреждений, общественных организаций и т.п.).

G3-технологии эффективно использовались в более 800 проектах национального масштаба.

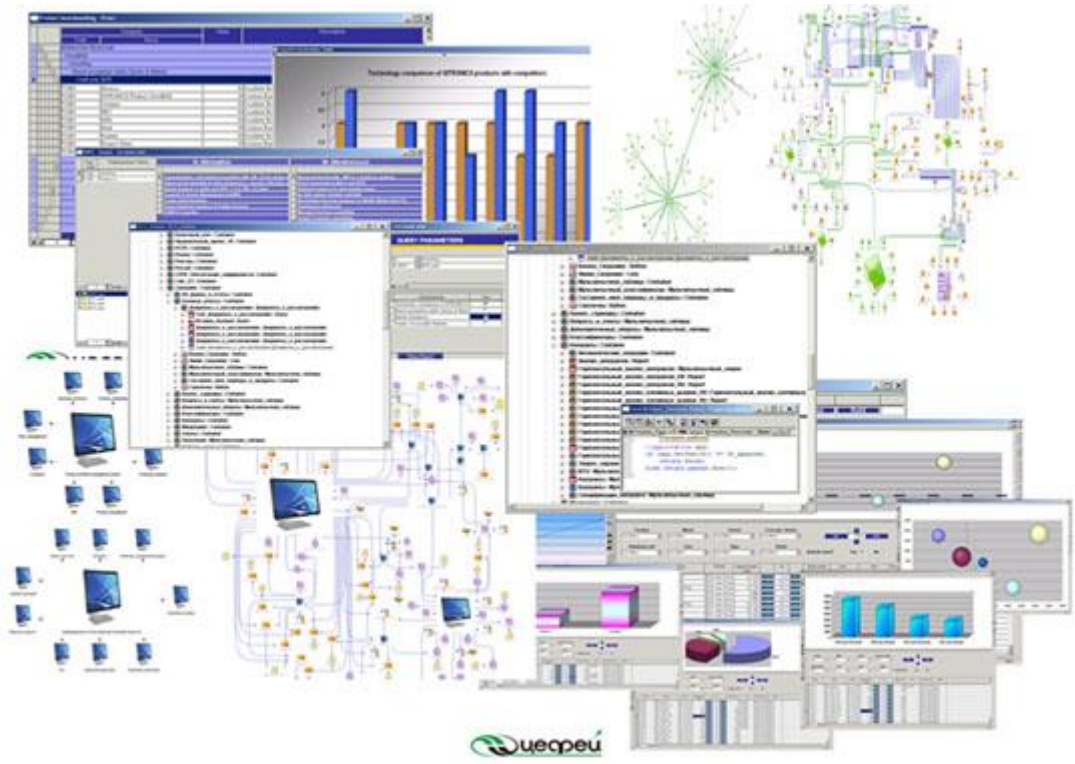
На основе реализованных проектов сформированы управленческие тренажёры – сетецентрические информационные G3-системы коллективного пользования с контрольными примерами для демонстрации и обучения навыкам работы в GGG (GRAPH) сетях нового поколения.

Наиболее популярны тренажёры: «G3-РОССИЯ», «G3-КОРПОРАЦИЯ».

В настоящее время **робот по программированию** успешно работает с гиперграфом - G3S, включающим:

сотни тысяч вершин (классы содержание, форма, поведение), более 2 миллиардов связей, около 100 000 взаимоувязанных таблиц базы данных (например, ORACLE), около 10 миллиардов записей, которые описывают знания об объектах и процессах управления проектов «G3-РОССИЯ», «G3-КОРПОРАЦИЯ».

На основании результатов экспертиз и отзывов международных организаций, структур NATO (RTO, SPS, NAMSA,..), ведущих институтов РАН и государственных корпораций можно утверждать, что предлагаемые технологии и системы не имеют мировых аналогов и на 5-10 лет превосходят уровень мировых фундаментальных исследований в этой области.



Увеличить >>>

G3-технологии обладают лицензионной чистотой и прошли регистрацию в Роспатенте. Ряд разработчиков удостоены Премии Правительства РФ в области науки и техники за исследование, разработку и внедрение в промышленность инновационных технологий Глобального Гносеологического Графа.

[1] Самоорганизующиеся системы (с противостоянием энтропийным разрушающим тенденциям) характеризуются наличием в системе активных взаимосвязанных элементов со способностью изменять свою структуру и формировать варианты поведения, сохраняя целостность и основные свойства (в технических и технологических системах изменение структуры, как правило, приводит к нарушению функционирования системы или даже к прекращению существования как таковой).

Адекватность сложной развивающейся самоорганизующейся модели (в отличие от привычного представления о математическом моделировании и прикладной математике) доказывается эволюционно по мере её коллективного формирования специалистами различных областей знаний в процессе познания объекта. В самоорганизующихся системах с ростом количества активных элементов не выполняется закономерность возрастания энтропии, а наблюдаются негэнтропийные тенденции, т. е. собственно самоорганизация.

Дополнительные материалы:

- Хохлова М.Н. Теория эволюционного моделирования, Москва,, 2004. — 60 с. ISBN 5-87911-115-6
- Волович И.В., Хохлова М.Н. О теории моделирования и гиперграфе классов // Труды Математического института им. В.А.Стеклова. — 2004. — № т. 245. — Р. 281-287.
- Хохлова М. Н. "Конец информационного общества. Новый Ренессанс", 2010
- Слюсар В. «Военная связь стран НАТО. Проблемы современных технологий». журн. «ЭЛЕКТРОНИКА: Наука, Технология, Бизнес» 4/2008
- Лупанов П. Е., Хохлова М. Н., Шиманов В. Л. "Анализ эффективности использования сетецентрической GGG-технологии на примере реализации проекта «G3-БЮДЖЕТ РФ», 2012
- Хохлова М. Н."SOA. СМЭВ. Электронный обмен или обман",
- Хохлова М. Н. «Гармогенез», электронное научное издание «Устойчивое инновационное развитие: проектирование и управление» www.rypravlenie.ru том 9No 2 (19), 2013, ст. 4 УДК 304.9, 330.11
- Хохлова М. Н. «Гармогенез. Поствинеровская кибернетика», «РУССКИЙ ИНЖЕНЕР», № 2(37)2013
- Хохлова М. Н. «Пришло время новой, поствинеровской кибернетики», «ПРОМЫШЛЕННИК РОССИИ», № 10/ 2010
- Хохлова М. Н. «Основы поствинеровской кибернетики», «ИНТЕГРАЛ», № 6 (56)/2010