Генезис графического пользовательского интерфейса. К теологии кода*

Михаил Куртов†

21 июня 2014 г.

Содержание

 Сокращенная и исправленная версия статьи: Куртов, Михаил. Блаженный Августин о графическом пользовательском интерфейсе / Искусство и наука как предполагаемое возможное. Сборник материалов международной художественной ассамблеи. Под. ред. Д. Булатова. Калининград: БФ ГЦСИ, 2014 (готовится к печати). Теория эволюции языков программирования кратко изложена в: Kurtov, Michael. L'évolution des langages de programmation à la lumière de l'allagmatique / Gilbert Simondon et l'invention du futur. Dir. par J-H. Barthélémy et V. Bontems. Paris: Klincksieck, 2014 (готовится к печати).

†К. филос. н. (СПбГУ). E-mail: kurtov@gmail.com

1 Введение

На первый взгляд графический пользовательский интерфейс кажется очень простой и тривиальной вещью. Его анализ показывает, что это — вещь, полная причуд, метафизических тонкостей и теологических ухищрений.

На графический пользовательский интерфейс (graphical user interface, GUI) общественные науки и философия обратили внимание в 1990-е гг. Сегодня существуют работы, исследующие его культурное[1], социальное[2], правовое[3], экономическое[4], психологическое[5], политическое[6], антропологическое[7] и пр. измерения. Однако до сих пор остаются нераскрытыми метафизические предпосылки его изобретения. Это тем более странно, что GUI опосредует наши отношения с миром и с другими людьми в большей степени, чем индустриальные медиа (в познании которых мы, как кажется, продвинулись достаточно далеко). Препятствием для философского исследования GUI является, на наш взгляд, отсутствие фундаментальных исследований по онтологии кода (языков программирования) — той реальности, которая обуславливает интерфейс и которую последний исторически и онтологически заслоняет.

Сам вопрос о связи между интерфейсом и кодом по большому счёту ещё не поставлен. Известны два типа рассказа о генезисе GUI — эргономический и культурологический. В первом авторы сосредотачиваются на прагматическим аспекте этого изобретения, помещая его в контекст исследований по психологии человеко-машинного взаимодействия[8]; во втором случае историю его рождения преподносят как результат культурных скрещиваний и мутаций (например, как пересечение поисков в области визуальной репрезентации и разработок по части компьютерной периферии[9]). В обоих случаях GUI предстаёт как интересная, богатая смыслами поверхность, за которой начинается «техническое, слишком техническое» — то, что, *по видимости*, не имеет прямого отношения к этим смыслам и чем поэтому можно пренебречь. Генезис и эволюция графического интерфейса остаются, таким образом, фактически оторванными в академическом сознании от генезиса и эволюции языков программирования, сделавших его

[1] Manovich, Lev. The Language of New Media. Cambridge, MA: MIT Press, 2001.

[2] Dourish, Paul. Where the Action Is: The Foundations of Embodied Interaction. Cambridge, MA: MIT Press, 2004.

[3] Lessig, Lawrence. Code And Other Laws of Cyberspace. New York: Basic Books, 1999.

[4] Aneesh, A. Virtual Migration: The Programming of Globalization. Durham: Duke University Press, 2006.

[5] Rogers, Yvonne; Sharp, Helen; Preece, Jenny. Interaction Design: Beyond Human. New York: John Wiley & Sons, 2011.

[6] Galloway, Alexander. Protocol: How Control Exists after Decentralization. Cambridge, MA: MIT Press, 2004.

[7] Suchman, Lucy. Human-Machine Reconfiguration: Plans and Situated Actions. NY and Cambridge, UK: Cambridge University Press, 2007.

[8] См, напр.: Grudin, Jonathan. A moving target: The evolution of human–computer interaction // Sears, Andrew; Jacko, Julie (eds.). Human-Computer Interaction Handbook. Boca Raton: CRC Press, 2007.

[9] Manovich, ibid.

возможным[10]. Возникают сомнения: да и есть ли вообще эта связь? а если есть, то что нам даст знание о ней? Но если даже это две реальности, существующие по своим законам, подобно физической и биологической, то наличие, скажем, биофизики намекает на то, что и в этом случае может потребоваться создание новой синтетической дисциплины.

Как нефилософы, так и многие философы сегодня разделяют убеждение, что знание внутреннего устройства технических объектов несущественно для повседневной жизни среди них: способы их использования прямо не зависят от этого знания, а уход за ними может быть передоверен третьей стороне (институту технической гарантии). Такое отношение утвердилось в эпоху индустриальных машин и породило социально-экономические структуры, детально проанализированные Марксом. Впрочем, для Маркса знание технических объектов также несущественно — куда важнее *обладание* ими: поскольку индустриальные средства производства включают в себя природные ресурсы, то всё можно свести к вопросу о неравномерности доступа к последним. Будучи само по себе нездоровым и отчуждающим, это отношение распространилось и на постиндустриальную, информационную технику. Однако постиндустриальные средства производства почти не связаны с природными ресурсами: экономически код доступен каждому.

Для нас важным различием между индустриальной и постиндустриальной техникой является то, что последняя задействует в качестве основного ресурса для своего развития не столько природу, сколько культуру[11]. Если граница между производством и потреблением индустриальных технических объектов имеет социально-экономический или природно-социальный характер, то в случае постиндустриальной техники это граница культурно-антропологическая: она проходит между *двумя образами культуры* и *двумя образами человека*. Это наша культурная инерция, культурное запаздывание заставляют думать о создании информационных технологий и об их использовании как о двух несвязанных реальностях (хотя исследователи не раз отмечали, что между программированием и использованием компьютера нет фундаментального различия[12]). Как образовалось это запаздывание? Ответить на этот вопрос можно только проследив генезис и эволюцию интерфейса и кода в их взаимосвязи.

Насколько нам известно, исследователи ещё ни разу не брались за подобную задачу: связать воедино интерфейс, код и культуру. Часть этой задачи — установление отношений между интерфейсом и культурой — выполняется сегодня культурологами и медиатеоретиками. Основной пробел, как уже было сказано, обнаруживается в философских исследованиях ко-

[10]См., напр.: Ess, Charles. Computer-mediated Communication and Human-Computer Interaction / Floridi, Luciano (ed.). The Blackwell Guide to the Philosophy of Computing and Information. New York: Wiley-Blackwell, 2003.

[11]Это вытекает из их онтологического различия: индустриальная техника по преимуществу энергетическая, тогда как постиндустриальная — информационная. Что было бы, если бы Маркс дожил до открытия возможности использовать электричество для передачи сигналов? Наша работа отчасти является попыткой ответить на этот вопрос.

[12]Ceruzzi, Paul. A History of Modern Computing. Cambridge, MA: The MIT Press, 2003. P. 81.

да. Общий изъян этих исследований состоит в том, что код берётся либо *не исторически*[13], т.е. вне логической связи между этапами его развития, либо *недостаточно технически*[14], т.е. вне единства и многообразия его деталей (чаще всего имеет место и то, и другое). Под историей кода, как правило, понимают что-то вроде естественной истории: развитие информационных технологий подобно отложению геологических слоёв, которые изучаются как данность[15]. Так происходит оттого, что исследователи мыслят код с помощью внешних ему средств — багажа антропологических, культурологических и пр. концепций, в которых они надеются найти ключ к этому новому явлению[16]. Но они тем самым продолжают неявно сохранять оппозицию между человеком и техникой, техникой и культурой: даже те философы, которые номинально придерживаются обратной позиции (выступают против «человеческой исключительности» и т.п.), всё равно тащат за собой ветошь представлений, сложившихся внутри этой оппозиции. Именно потому, что мы имеем дело с *сущностно новым явлением*, следует не *мыслить код* (поскольку сама современная философская мысль ему уже не современна), а *мыслить кодом* и *мыслить изнутри кода*. Общественные науки должны быть не столько средством для анализа (хотя код сам по себе, конечно, не предоставляет таких средств), сколько сами должны быть переопределены в результате его развёртывания[17].

В результате — бесшумный скандал: ни философы, ни специалисты по информатике до сих пор не смогли концептуализировать развитие информационных технологий. Новые изобретения возникают как бы случайно, без видимого порядка и логики; разработчики продвигаются вперёд на ощупь, без сознания собственных действий. У кода до сих пор нет *истории*, есть только *хронология*. У кода до сих пор нет своей *фундаментальной теории*, есть лишь попытки приладить к нему концепции из других наук (от математики и логики до социологии). Теория эволюции кода ещё не создана[18].

Ницше говорил, что философ должен быть филологом и психологом; сегодня философ должен быть историком и техником. Хочешь быть философом — пиши романы, говорил Камю; сегодня философ должен писать

[13] «Даже лучшие социологические исследования вычислительной техники игнорируют её историческую эволюцию, как если бы социологи наблюдали за тем, что дано, а не что быстро развивается» (Ceruzzi, ibid., p. 352).

[14] Это касается также и *software studies* — до недавнего времени переднего края гуманитарных исследований программного обеспечения; *недостаточно технично*, напр.: McKenzie, Adrian. Cutting Code: Software and Sociality. New York: Peter Lang, 2006).

[15] См., напр.: White, Graham. The Philosophy of Computer Languages / Floridi, Luciano (ed.). The Blackwell Guide to the Philosophy of Computing and Information, ibid.

[16] Яркий пример — Berry, David. The Philosophy of Software: Code and Mediation in the Digital Age. New York: Palgrave Macmill an, 2011.

[17] В данной статье мы, однако, установим лишь общую рамку, в которой это переопределение возможно.

[18] По крайней мере, такая теория — будь то имманентно-техническая, культурологическая или социологическая (в духе методологии social construction of technology) — нам неизвестна. Имеются лишь заявки на её создание; напр.: Gabriel, Richard. The End of History and the Last Programming Language / Patterns of Software: Tales from the Software Community. Oxford, New York: Oxford University Press, 1996.

код. Технические специалисты не создали теорию эволюции кода, поскольку им чужд историзм как метод познания, философы — поскольку они не способны преодолеть свою исходную, обусловленную самой ситуацией рождения философии нелюбовь к технике[19].

Пожалуй, единственным исследователем, в котором совмещалась любовь к sophia и к techne, был французский философ Жильбер Симондон, создавший в конце 1950-х гг. онтогенетическую теорию технических объектов. К сожалению, Симондон не оставил трудов по эволюции постиндустриальных технических объектов, *цифровых вещей*, однако им был разработан понятийный аппарат для определения сущности кибернетики как науки — *общая теория операции*. Широко опираясь в настоящей статье на эту теорию, мы, вместе с тем, столкнулись с ограничениями Симондоновой теории эволюции технических объектов, когда попытались приложить её к постиндустриальным объектам: неосязаемый, нематериальный характер последних, по всей видимости, выводит их из-под действия законов материальной эволюции. Отсюда наше неожиданное обращение к диалектике Гегеля (которого мы, впрочем, склонны прочитывать медиологически[20]).

Этой статьёй мы хотим изменить представление о *ближайшем* компьютера, графическом интерфейсе, показав его как *далёкое*. Проблема генезиса GUI станет для нас *точкой входа* в онтогенетическую теорию цифровых вещей. В заключение мы продемонстрируем возможности, которые предоставляет разработанный нами подход к анализу отношений между обществом–культурой и кодом.

2 Генезис графического пользовательского интерфейса

Под GUI обычно понимают графический пользовательский интерфейс с перекрывающимися (overlapping) окнами, разработанный лабораторией *Xerox PARC* в 1970-е гг. и впервые имплементированный в персональном компьютере *Xerox Alto* (1973). Его основой является концепция WIMP: окна (windows), иконки (icons), меню (menus), манипулятор (pointing device). GUI был популяризован благодаря его коммерческой реализации в операционной системе *Mac OS* (распространявшейся начиная с 1984 г. в персональном компьютере *Apple Macintosh*) и дальнейшей его эксплуатации компаниями *Commodore International*, *Atari*, *Microsoft* и др.

Как считается[21], лёгкость в освоении этого инструмента, опосредующего взаимодействие с компьютером, стала одной из главных причин взлёта популярности персональных компьютеров в 1980-е гг. и их массовой инкуль-

[19] Об этом парадигматическом для античной мысли конфликте между теорией как созерцанием и практикой как техникой см.: Фаррингтон, Бенджамен. Голова и рука в Древней Греции. Четыре очерка социальных связей мышления. СПб: СПбГУ, 2008.

[20] См.: Дебрэ, Режи. Введение в медиологию. М.: Праксис, 2010, с. 192.

[21] См. главу *GUIs* в: Stephenson, Neal. In the Beginning...was the Command Line. New York: William Morrow Paperbacks, 1999.

турации в 1990-е. Сегодня первый контакт с компьютером, как правило, начинается с контакта с GUI (хотя последний всё больше теснится сенсорным интерфейсом мобильных устройств — который, тем не менее, является его идеологическим преемником).

Графический пользовательский интерфейс есть этап в эволюции средств человеко-компьютерного взаимодействия (human-computer interaction, HCI). Эволюция HCI есть конвергенция двух эволюционных рядов — (I) *эволюции внешних средств* (периферия) и (II) *эволюции внутренних средств* (код). При этом *эволюция пользовательского интерфейса* сама включает в себя несколько этапов (III). Внешние средства мы будем называть компьютерными *медиа*, код и интерфейс — *метамедиа*[22].

I. В упрощённом виде эволюция периферии проходила следующие этапы[23]:

1) использование для ввода–вывода *перфокарт и перфолент*, ведущих своё происхождение от Жаккардовых ткацких станков; ввод мог также осуществляться при помощи тумблеров, вывод — при помощи ламповых индикаторов; перфокарты и перфоленты впоследствии сменяются магнитными лентами;

2) использование для ввода–вывода *телетайпа* (печатной машинки, передающей сообщения по электрическому каналу);

3) соединение телетайпа с *алфавитно-цифровым дисплеем* («glass Teletype») — в том числе с целью уменьшения расхода бумаги; возникновение пары клавиатура–дисплей;

4) создание *графического дисплея* (начало 1960-х гг.);

5) добавление к паре клавиатура–дисплей развитых *манипуляторов*, преобразующих движения пользователя в управляющие сигналы; наиболее распространённым из них сегодня является «мышь» (Д. Энгельбарт, 1968).

II. Эволюция внутренних средств есть эволюция информатики. Отметим её основные этапы:

1) *машинный код* — использование в качестве инструкций для ЭВМ операционных кодов (opcodes) в двоичном исчислении;

2) замена двоичных кодов мнемоническими буквенными обозначениями с целью упростить написание и чтение программ — появление языка *ассемблера* (assembly language);

3) разработка в 1950-е гг. так называемых *языков высокого уровня* (high-level languages) с целью ещё больше облегчить процесс программирования: языки высокого уровня отличаются тем, что они частично или полностью

[22]Под метамедиа мы будем понимать такие медиа, которые служат для создания других медиа. Понимание компьютера как метамедиума было предложено одним из пионеров информатики Аланом Кэем (см. ниже): «Он [компьютер] есть медиум, который способен динамически симулировать детали любого другого медиума, включая медиа, которые не могут существовать физически. Это не инструмент, хотя он может работать как многие инструменты. Это первый метамедиум, и как таковой он имеет столько степеней свободы для репрезентации и выражения, сколько ранее никогда не встречалось и которые едва ли было исследованы» (Kay, Alan. Computer Software // Scientific American, №251, 1984, p. 59).

[23]Подробнее см. в: Ceruzzi, ibid.

не зависят от технического субстрата (архитектуры ЭВМ) за счёт введения «человеческих» абстракций.

Об эволюции языков программирования высокого уровня речь пойдёт ниже.

III. Наконец, эволюция пользовательского интерфейса — также в упрощённом виде — проходит через три этапа:

1) *пакетный интерфейс* (batch interface) как конвергенция первого этапа эволюции внешних средств (перфокарты, перфолента) и первого этапа эволюции внутренних средств (машинный код);

2) *интерфейс командной строки* (command line interface, CLI) как конвергенция второго этапа эволюции внешних средств (телетайп) и второго и третьего этапов эволюции внутренних средств (ассемблер, языки высокого уровня); сюда же можно отнести так называемый текстовый пользовательский интерфейс (text user interface);

3) *графический интерфейс* как конвергенция пятого этапа эволюции внешних средств (клавиатура + графический дисплей + «мышь») и третьего этапа эволюции внутренних средств; появлению WIMP предшествует переизобретение полиэкрана и эксперименты с новыми манипуляторами (*Sketchpad*, *NLS*, *GRAIL*, etc.); разработка WIMP в лаборатории *Xerox PARC* опиралась на исследования в области зрительного восприятия (Ж. Пиаже, Л. Выготский, Дж. С. Брунер и др.).

Предварительным определением генезиса GUI поэтому является:

(1) $\mathbf{GUI}^{\mathrm{III}} = \mathbf{(клавиатура + дисплей + «мышь»)}^{\mathrm{I}} + \mathbf{(языки\ высокого\ уровня)}^{\mathrm{II}}$

В этом определении не хватает одного — уточнения относительно того этапа эволюции языков программирования, на котором они находились в момент изобретения GUI. Это уточнение является принципиальным, поскольку без определённых абстракций в языках высокого уровня, изобретённых не ранее 1960-х гг., не получил бы распространение и известный визуальный схематизм WIMP. Для того, чтобы внести это уточнение, необходимо пролить свет на сущность кода и его эволюцию.

3 Сущность кода

Важнейшим этапом эволюции компьютерных метамедиа стало пришедшее в начале 1950-х гг. осознание того факта, что вычисляющая и программирующая части компьютера могут быть объединены в одно целое (А. Гленни, Х. Рутисхаухер)[24]. С этого момента рождается компьютер как отдельная *вещь*, как индивид.

Программирование есть первичный (исторически и онтологически) способ взаимодействия с компьютером как с вещью. Наука об этом взаимодействии получила в 1959 г. название *computer science* (англ.) и *Informatik*

[24] Ceruzzi, ibid., p. 84.

(нем.). Традиционно информатику считают технической дисциплиной, вышедшей из лона математики. Мы, однако, полагаем, что она не является в чистом виде технической наукой и что её связь с математикой косвенная.

Классический учебник по программированию (SICP, 1985) начинается со следующего отрицательного определения:

> В основе нашего подхода к предмету лежит убеждение, что «компьютерная наука» [computer science] не является наукой и что её значение мало связано с компьютерами.

Далее разъясняется сущность этой не-науки:

> Компьютерная революция — это революция в том, как мы мыслим и как мы выражаем наши мысли. Сущность этих изменений состоит в появлении дисциплины, которую можно назвать компьютерной эпистемологией, — исследования структуры знания с императивной точки зрения, в противоположность более декларативной точке зрения классических математических дисциплин. Математика дает нам структуру, в которой мы можем точно описывать понятия типа «что такое». Вычислительная наука дает нам структуру, в которой мы можем точно описывать понятия типа «как».[25]

Соглашаясь с авторами этого хрестоматийного пассажа по части разведения математики и информатики как знания «что» и знания «как», мы не можем согласиться с отрицательным определением последней: информатика является наукой, но, возможно, в каком-то *доновоевропейском* смысле, а её историческое и культурное значение связано именно с компьютерами, но как с определёнными *вещами*.

Чтобы определить сущность кода, необходимо устранить его смешение с математическими сущностями. Исторически математика и информатика были связаны, но постепенно расходились. В культуре, между тем, продолжает жить их спутанный образ. Отчасти в этом повинно укрепившееся на начальном этапе представление о компьютере как о чём-то, что имеет отношение к числу: к такому пониманию подводят сами термины «электронно-вычислительная машина» и «цифровая техника». На деле, основой функционирования ЭВМ является не число и не цифра, а *чистое различие* между *1* и *0*, между бытием и ничто, между тварностью и нетварностью, выражаемое в машине при помощи электрического разряда и его отсутствия. Именно такой, философско-теологический смысл вкладывал Лейбниц в изобретённую им систему двоичного исчисления[26]. Само вычисление есть, по Лейбницу, какая-то деятельность ума ввиду этого различия.

[25] Абельсон, Харольд; Сассман, Джеральд Джей. Структура и интерпретация компьютерных программ. Москва: Добросвет, 2006. С. 16.

[26] Письмо к И.Х. Шуленбургу (29.03.1698). См.: Лейбниц, Готтфрид. Письма и эссе о китайской философии и двоичной системе исчисления. М.: Издательство ИФ РАН, 2005, с. 219.

При этом компьютер осуществляет эту деятельность ума *самостоятельно*. Компьютер — это, с одной стороны, вещь среди прочих вещей, с другой стороны, вещь действующая, вещь, которая движет и изменяет саму себя и другие вещи, — res computans, «вещь вычисляющая». Аристотель называл единичные вещи *субстанциями*; тогда компьютер есть *самодвижущаяся субстанция*, а computer science — первая современная (или, вернее, постсовременная) наука о самодвижущихся субстанциях. Если историческая роль метафизики заключалась в исследовании «что» вещей (*чтойности*, quidditas) и бытия этого «что», а роль математики — в исследовании количественной и пространственной структуры чтойности, то предметом информатики является «как» (*каковость*, quammitas) вещей.

Поскольку компьютер берёт на себя бремя механических вычислений, человеку остаётся лишь продумывать логику его поведения. Открытие этой *логики поведения* — названной в XX в. *алгоритмом* — и стало началом информатики[27].

Неформальное знание об алгоритмах существовало всегда (например, в виде составления кулинарного рецепта), однако формализация этого понятия началась только с возникновением первых проектов вычислительных машин. Самыми ранними и наиболее известными формальными дефинициями алгоритма являются «машина Тьюринга» (А. Тьюринг), «машина Поста» (Э. Пост), лямбда-исчисление (А. Чёрч). Важно правильно понять культурное значение этих формализмов: компьютер как вещь мог быть изобретён и *без них*. И фактически он и был изобретён без них: немецкий инженер Конрад Цузе создал первый работающий программируемый компьютер (*Z3*, 1941), не будучи знаком с трудами Тьюринга[28], которые стали отправной точкой для общемировой компьютерной науки (об изобретениях Цузе узнали позднее). Оба формализма скорее служат чем-то вроде *перевода*[29] с языка *чтойности* на язык *каковости*. Когда этот *перевод* был проделан, новая реальность, открытая компьютерами, связалась со старой: между современной и постсовременной культурами был переброшен мост[30].

Именно изобретение компьютера привело к легитимации научного знания о реальности «как». Впрочем, осмысление этой реальности на Западе началось задолго до трудов математиков и логиков — у авторов, которых иногда подводят под рубрику «философия становления» (Ф. Ницше, А.Н. Уайтхед и А. Бергсон). «Сущность» и «становление» — философские эквиваленты технических «что» и «как». Наиболее радикально и последо-

[27] На ранних этапах computer science определялась именно как «исследование алгоритмов» (Knuth, Donald. Selected Papers on Computer Science. Stanford, CA: Center for the Study of Language and Information, 1996, p. 5.).

[28] Zuse, Konrad. The Computer — My Life. Berlin: Springer Verlag, 1993. P. 53.

[29] Понятие *перевода* взято нами из социологии науки: «Перевод включает в себя создание конвергенций и гомологий через связывание [relating] вещей, которые до того были различными» (Callon, Michel. Struggles and Negotiations to Define what is Problematic and what is not: The Sociologic of Translation / Knorr, K.; Krohn, R.; Whitley R. (eds.). The Social Process of Scientific Investigation. Dordecht, Holland: D. Reidel Publishing Co., 1981, p. 211).

[30] Мы могли бы говорить о пост-модерне как современности в эпоху «машины Поста».

вательно различие между ними было проведено Анри Бергсоном в связи с проблемой времени. Человеческий интеллект, по Бергсону, не способен схватывать время как таковое, он может иметь дело только с отдельными, сколь угодно тонкими срезами времени, на которые первоначально и было направлено внимание метафизики и математики (лучше всего выразивших эту естественную установку интеллекта):

> *...мир, над которым оперирует математик, умирает и возрождается в каждый момент.* <...> Но, при таком понимании времени, как можно представить себе развитие, т.е. характерную черту жизни? Ведь развитие заключает реальную непрерывность прошлого в настоящем, заключает длительность, являющуюся *соединительной чертой*. Другими словами, познание живого существа или *естественная система* является познанием, относящимся и к промежуточному времени, тогда как знание искусственной или математической системы относится только к конечным пунктам.[31]

Иными словами, *как именно* в мире происходят изменения, математика — будучи инструментом интеллекта — не знает; она постоянно перепрыгивает через это «промежуточное время» ради установления статичных, неподвижных структур, являющихся «конечными пунктами» движения, но не самим движением. В этом смысле образ движения, который производится интеллектом, ничем сущностно не отличается от того, который производится дискретным механизмом компьютера как самодвижущейся субстанции. В обоих случаях этот образ будет лишь суммой состояний, которая никогда не равна движению. Время (или «длительность»), по Бергсону, постигается только через интуицию.

В работе Жильбера Симондона 1958 г. мы находим первое применение идей Бергсона к кибернетике — и эта же работа стала одной из первых попыток схватить новый феномен онтологически[32]. Симондон называет реальность «как» *операцией*, а реальность «что» — *структурой*. *Операция* и *структура* суть две фазы бытия, взаимно дополняющие друг друга, и, одновременно, два аспекта, два режима мысли, в которых внимание может быть направлено лишь на одну из фаз. *Операция* — это обращение одной *структуры* в другую, это то, что всегда «между» (metaxu) *структурами* (тогда как математика полагает, что между *структурами* может быть только другая *структура*, что сама *операция* есть не что иное, как просто ещё одна *структура*). Симондон подчёркивает, что, как и *структура, операция* имеет собственный *схематизм*.

Когда перед древнегреческим геометром стояла задача провести прямую, параллельную к данной, его внимание было направлено на *структуру* (заданную отношением параллельности), но сам жест прочерчивания прямой, *операция*, был несущественен, геометр от него абстрагировался. Таким

[31] Бергсон, Анри. Творческая эволюция. Материя и память. Мн.: Харвест, 1999. С. 36.

[32] Здесь и далее см.: Simondon, Gilbert. L'individuation à la lumière des notions de forme et d'information. Paris: Jérôme Millon, 2005, p. 559-566.

образом, всё здание современной, математизированной науки выстраивалось на приоритете *структуры* над *операцией* в указанном смысле; это же по большей части верно и для здания античной метафизики. И вместе с тем, этот жест прочерчивания — это и есть то, чем grosso modo занят программист: **приоритет *операции* над *структурой* составляет специфику информатики как науки**. Кибернетика, по Симондону, положила начало *общей теории операции* (*аллагматики* — от греч. *allattein*, «изменяться»). Можно сказать, что теоретическая деятельность в целом определяется преимущественным вниманием к *структуре*, практическая — преимущественным вниманием к *операции*. Математик и философ утверждают примат *созерцания* (theoria) над *действием* (praxis) и *деланием* (poiesis), программист — примат действия и делания над созерцанием.

Эта концептуализация призвана подвести философское основание под интуитивно понятные категории «что» и «как». Из неё следует, что информатика, хотя и была связана с математикой исторически, является скорее её *инверсией*, чем прямым развитием. Математика может создавать лишь различные *переводы* с языка *операции* на язык *структуры*, которые сами при этом будут *структурами* (как «машина Тьюринга» или соответствие Карри–Говарда). И наоборот: информатике известны не чистые *структуры*, но лишь бесконечные *операционные* приближения к ним (как, например, *доказательные вычисления*, computer-assisted proof). Заметим, что отсюда, возможно, берётся частая характеристика программистской деятельности как «магической»[33]: антропологически исследованием *операции* занимались не философия и наука, а искусство и религия (в основе искусства как techne, являющегося результатом poiesis, лежит представление об искусности, технической сделанности, а в основе религии — ритуал как особая практика, придающая верованию материальное воплощение).

Итак, **компьютер как самодвижущаяся субстанция есть вещь, способ существования которой покоится на приоритете *операции*.** Культурно-исторически этот приоритет был обусловлен тем, что информатика развивалась на основе уже упомянутого формализма Тьюринга: «машина Тьюринга» представляет собой бесконечную движущуюся ленту с ячейками, в которые управляющее устройство вписывает инструкции, в результате чего меняется состояние последнего. Движение ленты выступает аналогом изменения состояний сознания («state of mind»[34]). На базе формализма Тьюринга была создана наиболее распространённая сегодня архитектура персональных ЭВМ, носящая имя фон Неймана[35]. Однако факт изобретения компьютера независимо от этих формализмов (К. Цузе) указы-

[33] См., напр., лекцию Х. Абельсона на официальном канале MIT: «Мы можем назвать computer science инженерией или искусством. Но в действительности, как мы увидим, она имеет много общего с магией». URL: http://www.youtube.com/watch?v=2Op3QLzMgSY

[34] Turing, Alan. On computable numbers, with an application to the Entscheidungsproblem // Proceedings of the London Mathematical Society, Series 2, 42, 1936-1937, p. 250.

[35] Именно архитектуру фон Неймана часто делают ответственной за приоритет *операции* в репрезентации программ. См.: Backus, John. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. Communicatios of the ACM, vol. 21, N. 8, 1978.

вает на то, что дрейф от *структуры* к *операции* транскультурен и трансисторичен и охватывает собой, наряду с электронно-вычислительной техникой, прочие культурные явления (кинематограф, модернистскую живопись и музыку и т. д.).

Код мы определим как связывание *структур* и *операций* в новую *операцию*. Инструменты такого связывания называют *абстракциями* (связывание *отвлекает* от конкретной *структуры/операции*). Абстракции в информатике могут быть поняты как *модуляторы* (по Симондону, модулятор есть «активный ансамбль», устанавливающий отношение между *структурой* и *операцией*). Поскольку *операция*, в отличие от *структуры*, никогда не бывает полностью абстрактной, говорят о «конкретных абстракциях»[36].

В отличие от математики, абстрагирующее движение в коде ведёт не к отбрасыванию прежних уровней развития за счёт их включения в более общую *структуру*, а к их (временному) *сокрытию*[37]. Так образуются *слои абстракции* (abstraction layers). В той степени, в которой это позволяют выразительные средства конкретного языка, программист может перемещаться между этими слоями, т.е. связывать кристаллизовавшиеся *структуры* новыми *операциями*. Пользуясь терминологией Симондона, движение вверх по лестнице абстракций мы назовём *модуляцией*, вниз — *демодуляцией*. Это обратимое движение абстракций противоположно необратимому росту *структур* в математике, который Симондон называет *кристаллизацией*. Кристаллизация представляет собой *инверсию* модуляции[38] (демодуляция *реверсивна* модуляции).

Восходящее движение абстракций в коде можно рассмотреть также и *исторически* — как эволюцию представлений о компьютере как вещи.

4 Эволюция кода

Абстракции ничего не меняют в природе компьютера: они являются репрезентацией и одновременно реализацией различных способов его существования (аспектов функционирования). Абстракции, с одной стороны, существуют в уме программиста (как познавательные инструменты), с другой стороны, входят составной частью в исторически сложившиеся системы представлений, известные как языки программирования. Гносеологический аспект абстракций неотделим от онтологического: осуществляя модуляцию и демодуляцию сущностей внутри компьютера (и тем самым сущности компьютера), программист путешествует по истории кода[39].

[36] Hailperin, Max; Kaiser, Barbara; Knight, Karl. Concrete Abstractions: An Introduction to Computer Science Using Scheme. Boston: Course Technology, 1998.

[37] Colburn, Timothy. Philosophy and Computer Science. Armonk: Sharpe, 2000. P. 188.

[38] Simondon, ibid., p. 566.

[39] Такое историзующее понимание программистской деятельности (каждый слой абстракции есть не отбрасывание, но переход к иному эволюционному этапу развития кода) могло бы стать общим идеологическим решением проблемы *leaky abstraction* (Joel Spolsky): абстракции начинают «протекать», когда забывают об их истории (оригиналь-

Три указанных выше этапа эволюции языков программирования, — 1) машинный код, 2) ассемблер, 3) языки высокого уровня — суть три исторических способа репрезентации функционирования компьютера. С точки зрения результата работы компьютера на каждом из этих этапов ничего не меняется — трансформируется лишь человеческое представление о том, как коммуницировать с ЭВМ. Причём, как и в случае естественного языка, эти эволюционные этапы существуют диахронически и синхронически: они не отбрасываются, но продолжают синхронно существовать в виде скрытых слоёв абстракции.

Так, в трёх следующих примерах кода — каким бы он выглядел на каждом из этапов — совершается одно и то же действие: регистру процессора, называемому *AL*[40], присваивается число *128* (*10000000* — в двоичной системе, *80* — в шестнадцатеричной); или, говоря проще, в некий «ящичек» в памяти кладётся число *128* для какого-то его дальнейшего использования (например, для сложения с другим числом):

```
1) 10110000 10000000
2) MOV AL, 80h
3) _AL = 128;
```

Из этого примера видно, что для разных видов абстракции требовались разные способы символического выражения. Эволюция кода по сути имела не технический, а абстрактно-символический характер: технические науки всегда так или иначе задействуют природные ресурсы, тогда как единственным ресурсом для развития кода является культурный[41]. Если наш первый пример — это условное (при помощи символов нуля и единицы) представление триггеров в процессоре, то второй и третий примеры представляют собой отвлечение от этого технического субстрата при помощи лексем (MOV=*move* — мнемокод для операции копирования данных) и знаков пунктуации. Символические обозначения операций играют примерно ту же роль, что и математические обозначения: те и другие служат «сокращению работы мышления», по известному выражению Лейбница. Необходимо отметить ключевую роль языка ассемблера в этой эволюции: на его этапе цифра (сама являющееся символическим представлением дуализма бытия и ничто) становится буквой, тем самым связываются разные культурные миры, мир арифметики и мир грамматологии, «Афин» и «Иерусалима».

Слоистый характер абстракции распространяется и на понимание сущности программ: она интерпретируется в зависимости от выбранного для отсчёта слоя. При этом во всех случаях мы имеем дело с *двоякой* интерпретацией: с точки зрения пользователя, программа является файлом (тем,

---

ная проблема поставлена в: Kiczales, Gregor. Towards a New Model of Abstraction in Software Engineering. IMSA'92 Workshop on Reflection and Meta-level Architectures, 1992).

[40] *AL* — стандартное обозначение для «нижней», 8-битной части 16-битного регистра *AX* в архитектуре процессоров *Intel* серии *8086*.

[41] Мы опускаем здесь исследование зависимости software от hardware: разумеется, на первых этапах эволюции кода недоставало вычислительной мощности для реализации тех или иных программных идей, но уже с конца 1960-х гг. развитие hardware опережает развитие software (так называемый software crisis).

что занимает место в постоянной памяти) *и* процессом (тем, что занимает место в оперативной памяти), с точки зрения программиста — текстом *и* бинарным (скомпилированным, переведённым в двоичный код) файлом, с точки зрения теории императивного программирования — данными *и* процедурой, с точки зрения теории алгоритма — логикой *и* управлением[42] и т.д. Этот феномен получил название *дуальной природы* программ[43].

Общая теория *операции* позволяет дать объяснение этой дуальности: так на разных уровнях абстракции обнаруживают себя фазы *структуры* и *операции*.

Концептуализация отношения *структуры* и *операции* лежит в основе такого исторически важного для информатики понятия, как *парадигма программирования* (ПП)[44]. ПП является более высоким уровнем абстракции по отношению к языкам программирования; мы полагаем, что эволюцию языков программирования следует рассматривать именно с этого метауровня (условно говоря, на уровне *вида*, а не *индивида*).

ПП не имеет строгого определения, её понимают как «стиль программирования»[45], «модель вычислений»[46], общий подход к решению программистских задач. ПП есть уже какая-то абстракция работы hardware, поэтому это понятие, как правило, относят к языкам высокого уровня. Изобретающиеся языки программирования могут принадлежать к одной или сразу к нескольким парадигмам.

Мы определим *парадигму программирования* как общую репрезентацию дуализма *структуры* и *операции* при помощи базовой технической абстракции, которая соединяет их тем или иным способом. Заметим, что репрезентация всегда означает интерпретацию: то или иное решение, касающееся отношения *структуры* и *операции,* есть философское решение, а не «техническое»; собственно «техническим» решением является только выбор того или иного модулятора (абстракции, соединяющей обе фазы).

Исторически первыми и основополагающими парадигмами являются следующие три: *императивная*, *функциональная* и *объектно-ориентированная*.

Императивная парадигма (ИП) — это такая репрезентация работы компьютера, в которой его функционирование основано на приоритете внимания к *операции*. Термин «императивный» отражает приказательный характер коммуникации с машиной («выполни то-то и то-то»)[47]. В ИП основной

[42]Kowalski, Robert. Algorithm=Logic+Control // Communications of the ACM 22 (7), 1979. P. 424-436.

[43]Colburn, ibid. P. 199.

[44]Понятие введено Р. Флойдом в 1979 г. по аналогии с понятием парадигмы у Т. Куна, которое, правда, не имеет с первым практически ничего общего: Floyd, Robert W. The Paradigms of Programming // Communications of the ACM, 22(8), 1979, p. 455-460.

[45]Watt, David A. Programming Language Design Concepts. New York: John Wiley & Sons, 2004. P. 5.

[46]Van Roy, Peter; Haridi, Seif. Concepts, Techniques, and Models of Computer Programming. Cambridge, MA: The MIT Press, 2004. P. xiii.

[47]Историческую первичность императивного, т.е. *операционного* мышления в информатике и исторический приоритет *структурного* мышления в науке можно объяснить

технической абстракцией, интерпретирующей дуализм *структуры* и *операции*, выступает понятие *переменной*: программа представлена в сознании программиста как последовательная смена состояний (states) за счёт изменения значения переменных (при помощи операторов присваивания). На начальном этапе (ещё до введения понятия парадигмы) даже бытовало мнение, что тот, кто понял, как работать с переменными, понял «квинтэссенцию программирования» (Э. Дейкстра)[48]. ИП неслучайно основана на приоритете операции: абстракция работы компьютера в ней минимальна, она фактически повторяет логику hardware. Эта парадигма представлена такими исторически первыми языками, как Plankalkül (1945), Flow Diagrams (1946), Short Code (1950), Speedcoding (1953), FORTRAN (1957), ALGOL (1958) и др.[49]

Исторически второй парадигмой является *функциональная* (ФП). Она возникла под влиянием математических идей (лямбда-исчисление А. Чёрча) и потому репрезентирует работу компьютера с точки зрения приоритета *структуры*. Базовой технической абстракцией в ней является понятие функции в смысле, близком к математическому. В функциональных языках программирования, таких как Composition (Haskell Curry, 1948) или LISP (John McCarthy, 1958), нет смены *состояний*, а данные являются неизменяемыми (переменной можно присвоить значение лишь единожды). Программист не заботится о порядке выполнения (control flow) и видит программу как статичную систему отображений, через которую только в момент выполнения пропускается *операция*, оживляя её (т.е., несмотря на приоритет *структ*уры в ФП, выполнение программы утверждает приоритет *операции*).

Позднее (конец 1970-х гг.) было введено понятие *декларативного* программирования для обобщения сущности неимперативных языков — функциональных, логических и пр.; однако, поскольку нас интересует именно историческое развитие, т.е. эволюция представлений, мы будем здесь противопоставлять императивную парадигму функциональной.

Интересная особенность обеих парадигм: хотя переменная существует как *операционный* орган, она отсылает к значению, т.е. к *структуре*; и хотя функция существует как *структурный* орган, она отсылает к функционированию, т.е. к *операции*. Это согласуется с наблюдением Симондона, что в системе, привилегирующей *операцию*, «действие становится синонимом

социально-антропологически: источником того и другого является сохраняющаяся и по сей день антропологическая диспозиция, в которой отношение к технике в целом является «рабовладельческим». Как Античная метафизика презирала ремёсла (т.е. области знаний об *операции*), так и научная мысль XX в. смогла признать измерение *операции*, лишь поместив её в вертикальную структуру управления и подчинения. Об этой диспозиции см.: Simondon, Gilbert. Du mode d'existence des objets techniques. Paris: Aubier, 1989, p. 9-19. Об античном отношении к технике см: Фаррингтон, ibid.

[48]Dijkstra, Edsger W. Notes on Structured Programming / Dahl, Ole-Johan; Dijkstra, Edsger W.; Hoare, C.A.R. Structured Programming. London: Academic Press, 1972. P. 11.

[49]Подробнее см. в : Knuth, Donald; Pardo, Trabb. The Early Development of Programming Languages / A History of Computing in the Twentieth Century. New York: Academic Press, 1980.

опространствливания, обездвижения, структурирования», и наоборот[50].

Императивная и функциональная парадигмы — два крайних взгляда, две полярных репрезентации дуализма *операция–структура*. Одну можно назвать инверсией другой. Попыткой синтезировать эти взгляды стала третья парадигма — *объектно-ориентированная* (ООП).

Объектом называется изобретённая в середине 1960-х гг. техническая абстракция, в которой *структура* и *операция* объединены в одно целое (или, по крайней мере, которая мыслится как претензия на подобный монизм). В первой публикации об ООП, адресованной широкой аудитории, говорилось:

> Традиционное представление о программных системах заключается в том, что они состоят из коллекции данных, представляющей какую-то информацию, и набора процедур для манипуляций с этими данными... Проблематичным в этом взгляде на программы как данные/процедуры является то, что с данными и процедурами обращаются так, как если бы они были независимы друг от друга, тогда как в действительности дело обстоит наоборот... Вместо двух типов сущности, представляющих информацию и манипуляцию с ней независимо друг от друга, в объектно-ориентированной системе имеется один-единственный тип сущности, представляющей оба типа.[51]

Впервые эта абстракция была имплементирована в языках Simula 67 и Smalltalk. Технически объект — это сущность, которая имеет определённые атрибуты и поведение (т.е. включает в себя и *структуру*, и *операцию*) и способна взаимодействовать с другими объектами. Принято считать, что ООП *ортогональна* ИП и ФП, что означает, что две последние могут сочетаться с первой (скажем, более императивный вариант ООП представлен языками C++ и Java, а более функциональный — языком Smalltalk; при этом в императивных и функциональных языках могут использоваться инструменты ООП — например, Objective-C и CLOS соответственно).

Практическим мотивом к разработке понятия объекта стало стремление уменьшить сложность репрезентации (complexity management): в результате объединения *структуры* и *операции* в одной сущности (так называемая *инкапсуляция*) скрывается неактуальная информация (information hiding) и разгружается внимание программиста. Теоретическим мотивом стало осознание ограниченности репрезентации функционирования ЭВМ в модусе только *структуры* или только *операции*. Создатель термина «объект», соразработчик языка Smalltalk и сотрудник лаборатории *Xerox PARC* Алан Кэй писал:

> В компьютерных терминах Smalltalk — это рекурсия понятия самого компьютера. Вместо разделения «компьютерной всячи-

[50] Simondon, L'individuation..., p. 565.

[51] Robson, David. Object-Oriented Software Systems // BYTE Magazine, August 1981. P. 76.

> ны» на вещи, каждая из которых менее сильна, чем целое, — например структуры данных, процедуры и функции, эти обычные параферналии языков программирования, — объект Smalltalk'а являет собой рекурсию полных возможностей компьютера.[52]

Не будет преувеличением сказать, что изобретение объекта имело метафизическую подоплёку: природа компьютера дуальна, однако сам компьютер как вещь — един; почему бы не попытаться истолковать этот дуализм монистически?

Проиллюстрировать различия в репрезентации функционирования компьютера в трёх парадигмах можно следующим примером. Представим, что есть три ангела (или, говоря более программистским языком, демона), которые ответственны за обеспечение вращения Земли вокруг собственной оси в течение невисокосного года. Как именно вращать Землю, они не знают, это знание им передают другие ангелы. Императивный ангел сперва создаёт переменную и присваивает ей значение *0*; затем 1) вращает Землю, 2) увеличивает значение переменной на единицу, 3) проверяет, равно ли это значение *365*; если не равно, тогда он повторяет последние три шага (если равно — празднует Новый год). Функциональный ангел объявляет о существовании множества (в математическом смысле), состоящего из *365* элементов, а затем каждому из элементов этого множества ставит в соответствие операцию вращения Земли. Наконец, объектный ангел посылает числу *365* (которое является объектом) сообщение «повторяй» с указанием на операцию вращения Земли.

Эти алгоритмы могут быть реализованы в коде следующим образом (использованы только языки, принадлежащие к «чистым» ИП, ФП и ООП соответственно):

FORTRAN 77:

```
INTEGER I
I = 0
10 ROTATE
I = I + 1
IF (I.NE.365) THEN GOTO 10
```

Haskell:

```
x = [0..364]
main = do
     mapM_ rotateEarth x
```

Smalltalk 80:

```
365 timesRepeat: [rotateEarth].
```

[52] Kay, Alan C. The Early History Of Smalltalk // ACM SIGNPLAN Notices, Vol. 28, No. 3 March 1993. P. 3.

Социокультурное значение ООП огромно. Часто говорят, что она стала началом индустриализации программной разработки и главным фактором в преодолении так называемого software crisis (длившегося примерно с конца 1960-х по середину 1980-х)[53]. Подавляющее большинство создаваемых сегодня программ написаны на объектно-ориентированных языках.

В нашем случае первостепенный интерес имеет следующее обстоятельство: среда разработки первого полностью объектно-ориентированного языка Smalltalk, созданная на базе уже упомянутого компьютера *Xerox Alto*, и была той программой, где был *впервые* реализован GUI (как мы его определили выше). В ходе её разработки оказалось, что элементы графического интерфейса — в частности, «окно» — наиболее органично описываются при помощи понятия объекта: «окно» имеет какие-то атрибуты (заголовок, размер, место на экране, статус: активно/не активно и пр.), т.е. *структуры*, и поведение (его можно свернуть, развернуть, переместить и пр.), т.е. *операции*. «Окно» — это часть пространства, визуально репрезентирующая объект. Если программа имеет многооконный интерфейс (multiple document interface) или многодокументный интерфейс с вкладками (tabbed document interface), то «окна» репрезентируют, кроме того, иерархические отношения между объектами. Изобретение GUI исторически неотделимо от изобретения понятия объекта (хотя последний возникает раньше).

История коммерциализации и инкультурации GUI, начинавшегося как чисто экспериментальная разработка, неоднократно освещалась в литературе: в 1979 г. Стив Джобс посещает лабораторию *Xerox PARC,* где Алан Кэй и его коллеги демонстрируют ему среду языка Smalltalk, после чего Джобс заимствует идею GUI при разработке системы *Mac OS*. ООП широко распространяется вместе с GUI как наиболее естественный инструмент для его реализации. Сегодня GUI полностью интегрирован с ООП-технологией: за исключением каких-то экзотических случаев все элементы графического интерфейса, с которыми сталкиваются пользователи, представлены в коде как объекты (да и вся программа, как правило, представляет собой один сложно структурированный объект).

Теперь мы наконец готовы уточнить генезис графического пользовательского интерфейса:

$$(2)\ \mathbf{GUI}^{\mathrm{III}} = (\mathbf{клавиатура} + \mathbf{дисплей} + \mathbf{«мышь»})^{\mathrm{I}} + (\mathbf{ООП})^{\mathrm{II}}$$

Однако и этого уточнения ещё недостаточно. За изобретением GUI стояла более конкретная абстракция объектно-ориентированной технологии, культурно-историческое значение которой пока что не оценено и не изучено[54]. Для того, чтобы внести это последнее уточнение, проясним культурно-философский смысл понятия объекта.

[53] См.: Cox, Brad J. There Is a Silver Bullet // BYTE Magazine, October 1990.

[54] О культурном значении ООП в целом писал, скажем, один из её создателей Оле-Йохан Даль. См.: Dahl, Ole-Johan. The Birth of Object Orientation: the Simula Languages / Owe, Olaf; Krogdahl, Stein; Lyche, Tom (eds.). From Object-Orientation to Formal Methods. Essays in Memory of Ole-Johan Dahl. New York: Springer, 2004.

5 Философия кода

В изобретении объекта рефлексия постиндустриальных технологий, возможно, впервые явно соединилась с историей метафизической мысли. Если один из создателей языка Simula Оле-Йохан Даль ещё двигался по пути расширения структур данных посредством включения в них процедур (следуя общему для эпохи вектору декомпозиции и модуляризации)[55], т.е. в целом ещё находился в рамках императивного мышления, то Алану Кэю принадлежит осмысление объекта как сущностно нового явления и нового этапа эволюции абстракций. В своих мемуарах он так писал о языке Smalltalk:

> Его способ создания объектов довольно платонистичен в том, как некоторые из них функционируют в качестве идеализации концептов — Идей, — из которых могут быть созданы манифестации. То, что сами Идеи суть манифестации (Идеи Идеи) и что Идея-Идея есть род-Манифестации-Идеи [kind-of-Idea-Manifestation], — что есть род себя же самого, — так что система полностью самоописательна, это Платон расценил бы как чрезвычайно практичную шутку.[56]

Прежде, чем двигаться дальше, сделаем набросок основных технологических принципов, которые следуют из понятия объекта. По поводу последних в литературе существует большое количество толков, за которыми стоят разные метафизические установки. История истолкований понятия объекта сама по себе являет собой отдельную социокультурную интригу, или, говоря на программистском сленге, летопись holy war. Его интерпретация в таких языках, как C++ и Java, направила культурное развитие в определённое русло (например, структура и организация современных software-компаний в значительной степени определены этим истолкованием: оно располагает к громоздким энтерпрайз-решениям и привлечению среднеквалифицированных сотрудников[57], а наиболее популярные программные продукты, созданные при помощи этих языков, часто имеют соответствующую славу из-за недостатков в дизайне последних[58]), что должно стать предметом специального исследования[59].

[55] Для Оле-Йохана Даля это изобретение есть в первую очередь спецификация структур данных и затем определение операций с ними (Dahl, Ole-Johan; Hoare, C.A.R. Hierarchical Program Structures / Dahl; Dijkstra; Hoare. Structured Programming, ibid., p. 202).

[56] Kay, ibid., p. 3.

[57] См.: Graham, Paul. Why ARC isn't especially Object–Oriented. 2009. URL: http://www.paulgraham.com/noop.html

[58] Kay, Alan. Programming and Scaling (video). 2011. URL: http://www.teletask.de/de/archive/lecture/overview/5819/

[59] Политическую интерпретацию этим технологиям попытался дать А. Гэллоуэй; однако его работа имеет перечисленные выше недостатки: код в ней мыслится изнутри современных политических концепций (часть *Badiou and Java*), тогда как следует мыслить политическое изнутри кода. См.: Galloway, Alexander. The Poverty of Philosophy: Realism and Post-Fordism. 2012. URL: http://www.jstor.org/stable/10.1086/668529

Согласно наиболее распространённой в индустрии интерпретации ООП-технологии[60], последняя покоится на трёх механизмах — *инкапсуляции*, *полиморфизме* и *наследовании*. Эти механизмы могут имплементироваться раздельно (скажем, *модульное* программирование также прибегает к инкапсуляции), поэтому важно их сочетание. Под инкапсуляцией понимается сокрытие и связывание *структур* и *операций* внутри объекта: это уменьшает количество сущностей, с которыми имеет дело программист. Полиморфизм означает, что одно и то же имя объекта используется для разных, но семантически схожих объектов (имя есть «двойственный символ»[61]) — тем самым достигается удобство в обращении с разными типами данных через единый интерфейс (например, оператор «+» может означать как арифметическое сложение, так и конкатенацию, т.е. склейку символов). Это помогает сосредотачиваться на семантике операций и абстрагироваться от их технической реализации. Наконец, наследование означает, что объекты наследуют или передают в наследство некоторые или все свои *структуры* и *операции* другим объектам, в результате чего выстраивается *иерархия классов* (*класс* есть абстрактное описание, или *тип*, объекта). Наследование позволяет прибегать к *повторному использованию* кода (code reuse), что экономит время разработки.

Вернёмся к высказыванию Кэя. Его сравнение объекта с платоновской Идеей не единично. В работах по computer science сходство абстракций ООП с философемами античной классики — аристотелевских «вторичных субстанций» с *классами*, платоновской Идеи (Архетипа) с *прототипом* (первичный объект, служащий для генерации других объектов в так называемом *прототипном* стиле ООП), и пр., — является общим местом. Причём эта аналогия может быть прослежена вплоть до деталей: так, механизм полиморфизма повторяет проблематизацию «одноименности» и «соименности» (унивокации и эквивокации, говоря языком схоластики) у Аристотеля, а механизм наследования объектов (иерархия классов) порождает системы родовидовых классификаций, подобные аристотелевским[62].

Более того: сам генезис платоновской Идеи подобен генезису объекта в информатике. Досократическая мысль была одержима вопросом о движении. Два крайних ответа на этот вопрос, как считается, были даны Гераклитом и Парменидом: первый утверждал, что бытие есть движение и изменение («всё течёт»), а второй — что бытие неизменяемо, поскольку движение логически невозможно (бытие подобно «глыбе прекруглого шара»). Изоб-

[60] См.: Sebesta, Robert W. Concepts of Programming Languages. New Jersey: Pearson Education, 2006.

[61] Определение, данное в оригинальной статье 1967 г.: Strachey, Christopher. Fundamental Concepts in Programming Languages // Higher-Order and Symbolic Computation, №13, 2000, p. 35.

[62] См.: Hsu, Hansen. Connections between the Software Crisis and Object-Oriented Programming. URL: http://www.sigcis.org/files/Hsu – Software Crisis and OOP.pdf. Rayside, Derek; Campbell, Gerard T. Aristotle and Object-Oriented Programming: Why Modern Students Need Traditional Logic. URL: http://dis.eafit.edu.co/depto/documentos/p237-rayside - Aristotle and Object-Oriented Programming. Why Modern Students Need Traditional Logic.pdf

ретение Платоном Идеи стало решением дилеммы Парменида–Гераклита (причём больше клонящимся в сторону парменидова варианта): Идеи выступили как неизменяемое, вечное, не имеющее «состояний» бытие, как модель или Форма, а вещи — как изменения ввиду и относительно этих Идей, как проекции модели. В дальнейшем Аристотель развил платоновскую идею Идеи, связав понятия изменяемости и неизменности с понятиями материи и формы. Идея как «вид» (idea этимологически связана с video) есть то, что возникает не в уме, а в умозрении (в отличие от натурфилософских первовеществ, данных только в уме, т.е. не обладающих формой, — таких как апейрон, огонь или число); так же и объект есть данная в уме абстракция, которая при этом первой из всех абстракций в истории информатики обрела зримость в качестве элементов GUI.

Налицо, таким образом, не только сходство объекта с Идеей, но и сходство императивной и функциональной парадигм с мыслью Гераклита и Парменида соответственно: философемы этих досократиков подобны технологемам ИП и ФП, философемы Платона и Аристотеля — технологемам ООП. Отсюда можно сделать предварительный вывод, что эволюция абстракций в философии проходит те же этапы, что эволюция абстракций в информатике. Поскольку, как было сказано выше, античная метафизика зиждется на приоритете *структуры* (сама Идея как первое собственно философское изобретение утверждает приоритет *структуры*), а информатика — на приоритете *операции*, то их эволюции являются как бы *инверсиями* друг друга: там, где в метафизике *структура*, в информатике *операция*, и наоборот. Отличием информатики от метафизики является приоритет *операции*, а также более сжатый и краткий характер её эволюции.

Может показаться странным, что те изобретения, которые делались, казалось бы, чисто случайно, на основе сугубо практических нужд (в стремлении убавить сложность репрезентации), выстраиваются в определённую эволюционную линию. Но помня о том, что *операция* имеет свой собственный *схематизм*, мы можем трактовать computer scientists не столько как инженеров (в индустриальном смысле), сколько как *теоретиков практики*.

Теорию, согласно которой развитие *операционной* мысли в культуре следует за развитием *структурной* мысли и в сжатом кратком инверсированном виде повторяет его основные этапы, мы назовём *гипотезой операто́рной рекапитуляции* (другая формулировка гипотезы: эволюция computer science есть сжатый инверсированный эквивалент эволюции онто-теологии). Поскольку информатика есть двигатель постиндустриальной культуры, более «сильным», расширительным вариантом этой гипотезы был бы следующий: постиндустриальная, постсовременная культура — начавшаяся после «заката Европы», «конца истории» и «конца метафизики», — проходит те же этапы развития, что и западноевропейская культура в целом, начиная с Античности. Период 1900–1940-х гг. был периодом коллизии культурных «плит» и переходом к новому культурному циклу. Исчерпав свои внутренние потенции, западная культура как будто свернулась в одной-единственной вещи, дублирующей человеческие

способности, и развитие информационных технологий есть не что иное, как её онтологическая развёртка.

Выдвижение этой гипотезы ставит как минимум два вопроса. Во-первых, в каких культурно-исторических механизмах она находит себе обоснование? Во-вторых, можно ли, ввиду этой гипотезы проследить дальнейшую эволюцию информатики?

Первый вопрос отсылает к проблематике генезиса и развития культур, впервые очерченной Освальдом Шпенглером (1918). Шпенглер анализирует культуры отталкиваясь от понятия *гомологии*, заимствованного из биологических наук (у зоолога и палеонтолога Р. Оуэна), где оно означает морфологическую эквивалентность в противоположность аналогии как эквивалентности функций: «гомологичны легкое наземных животных и плавательный пузырь рыб, аналогичны — в смысле употребления — легкое и жабры»[63]. При переносе этого понятия на анализ культур отношения между их частями перестраиваются:

> Гомологичными образованиями являются... античная пластика и западная инструментальная музыка, пирамиды 4-й династии и готические соборы, индийский буддизм и римский стоицизм (буддизм и христианство даже не аналогичны), эпохи «борющихся уделов» Китая, гиксосов и Пунических войн, Перикла и Омейядов, эпохи Ригведы, Плотина и Данте. Гомологичны дионисическое течение и Ренессанс, аналогичны дионисическое течение и Реформация[64].

Гомологию как познавательный метод можно поставить в один ряд с «синхронистичностью» К.Г. Юнга, «квази-причинностью» Ж. Делёза и другими попытками схватить акаузальную связь между явлениями.

Различение *аналогии* и *гомологии* существенно для нас, поскольку мы полагаем, что истинный характер отношений между абстракциями в метафизике и информатике не аналогический (как вскользь было замечено выше), а *гомологический*: они выполняют в культуре разные функции, но занимают структурно соответствующие позиции. Это структурное соответствие Шпенглер схватил в термине «одновременность»:

> Я называю «одновременными» два исторических факта, которые выступают, каждый в своей культуре, в строго одинаковом — относительном — положении и, значит, имеют строго соответствующее значение. Было уже показано, что развитие античной и западной математики протекает в полной согласованности. Здесь, стало быть, позволительно было бы назвать Пифагора и Декарта, Архита и Лапласа, Архимеда и Гаусса одновременными. Одновременно протекает возникновение ионики и барокко.

[63] Шпенглер, Освальд. Закат Европы. Том 1. М.: Мысль, 1993. С. 270.
[64] Ibid., с. 271.

> Полигнот и Рембрандт, Поликлет и Бах — современники. Одновременными предстают во всех культурах Реформация, пуританизм, прежде всего поворот к цивилизации.[65]

Понятие «одновременности» (которое мы бы определили как трансисторическое тождество исторически различного) в том или ином виде стало краеугольным камнем для всех последующих теорий социокультурных циклов (П. Сорокин, А. Вебер, А. Тойнби и др.).

Теперь мы можем сказать, что абстракции в метафизике *гомологичны* абстракциям в информатике, а объект «одновременен» платоновской Идее, т.е. имеет для постиндустриальной культуры то же значение. Это утверждение не будет казаться столь странным, если думать о компьютерных программах не как о «чистых» технических объектах, а как о формах мысли. Компьютерная программа — это в первую очередь *не как сделать что-то*, а *как думать о чём-то*. Если взор античных философов был обращён на мир как на неподвижную или движимую субстанцию, то информатика — это мысль о *самодвижущейся* субстанции. Тогда допустимо будет рассматривать эволюцию абстракций в информатике как подчиняющуюся диалектическим законам *движения рефлексии* у Гегеля. Отсюда мы можем перейти к ответу на второй вопрос, поставленный в связи с нашей гипотезой.

Значение философии Гегеля состоит для нас в том, что она была первой завершённой попыткой показать, что «история философии не есть вслепую набранная коллекция *взбредших в голову мыслей*, ни *случайное* движение вперед», но, наоборот, «необходимое возникновение философских учений друг из друга, так что каждое из них непременно предполагает предыдущее»[66]. *Движение рефлексии* у Гегеля — это и *субъективный* процесс мышления, и *объективное* развёртывание мышления в *истории*, регистрируемое философией и схватываемое в воспоминании: в истории *дух* осуществляет *рефлексию себя* (точнее, дух есть имя для этого осуществления).

Эволюция кода есть *саморефлексия* компьютера как самодвижущейся субстанции. *Двойное* движение модуляции и демодуляции есть то, что отличает информатику от математики и сближает её с метафизикой[67]. Платоновское понятие Идеи было первым моментом саморефлексии в истории философии, понятие объекта — в истории информатики (объект есть «рекурсия понятия самого компьютера», согласно Кэю).

Симондон утверждал (в противовес «философам субстанции», вроде Платона или Декарта, или «философам становления», типа Гераклита или

[65] Ibid.

[66] Гегель, Г.В.Ф. Лекции по истории философии. Книга третья. СПб: Наука, 1994. С. 571.

[67] «В философском познавании становление наличного бытия как *наличного бытия* также отличается от становления *сущности* или внутренней природы дела. Но философское познавание, во-первых, содержит и то и другое, тогда как математическое познавание, напротив, изображает только становление *наличного бытия,* т.е. *бытия* природы дела в *познавании* как таковом. Во-вторых, философское познавание объединяет и эти два особых движения» (Гегель, Г.В.Ф. Феноменология духа. СПб: Наука, 1994, с. 22).

Ницше), что ни *структура*, ни *операция* не могут быть абсолютизированы: «эпистемологический монизм структуры и операции не остаётся верен сам себе и воссоздаёт в ходе своего развития тот термин, который он первоначально исключил»[68]. Иначе говоря, если даже на каком-то этапе возникает синтез *структуры* и *операции*, в дальнейшем выясняется, что этот синтез был неуравновешен и клонился в сторону той или другой фазы, после чего синтетическое понятие дополняется противоположным ему. Таким образом, имеет место исследованный Гегелем диалектический процесс отрицания и отрицания отрицания (*снятия*): *полагающая* рефлексия *структуры/операции* («тезис») входит в противоречие с их *внешней* рефлексией («антитезис»), которая «снимается» *определяющей* рефлексией («синтез»), — система выходит на более высокий уровень абстракции, и диалектическое движение начинает повторяться.

Так, уже в 1980-х гг. в программистской среде возникают сомнения в том, что понятие объекта пригодно для симуляции реального мира, который якобы состоит скорее из отношений, нежели из объектов. В итоге объект, первоначально будучи претензией на монизм, на синтез *структуры* и *операции* (т.е. сам будучи *определяющей* рефлексией *структуры* и *операции*), эволюционно занял место *структуры*, а комплементарная ей *операция* была воссоздана в виде понятия *отношения* (relation) между объектами. На этом новом этапе эволюции абстракций объект стал *полагающей* рефлексией, а понятие *отношения* — его *внешней* рефлексией.

Программистский скепсис относительно объекта, возможно, был связан с искажением изначальных идей ООП в таких языках, как C++ или Java, где объекты интерпретируются скорее как некая расширенная структура данных и являются, по сути, просто прибавкой к императивной идеологии. В чисто объектно-ориентированном языке Smalltalk понятие объекта было сбалансировано понятием *посылки сообщений* (messaging), поэтому язык не нуждался в специальном, несинтаксическом выражении отношений. Для разъяснения сущности messaging Кэй обратился к одному из ключевых понятий дзен-буддизма, *ma,* — которое удивительным образом совпадает по смыслу с *операцией*:

> Мне жаль, что я когда-то ввел в употребление для этого предмета термин «объекты», потому что из-за него много людей сфокусировались на менее важной идее. Более важная идея — это «посылка сообщений»: это то, что является ядром Smalltalk/Squeak... У японцев есть словечко — ma — для обозначения «того, что между»... Ключ к созданию отличных и способных к росту систем — это в большей степени разработка того, как их модули коммуницируют друг с другом, чем того, какими должны быть их внутренние свойства и поведения. Подумайте об Интернете — чтобы жить, он (а) должен позволять существовать разным типам идей и реализаций, выходящим за пределами какого-либо единого стандарта, и (б) позволять существовать варьирующим-

[68] Simondon, ibid., p. 564.

> ся степеням допустимого взаимодействия [interoperability] между этими идеями[69].

В начале 1990-х создаются специальные языки графического описания для формализации *отношений* посредством *концептуальных схем* (conceptual schema), выражаемых с помощью диаграмм (диаграммы состояний, диаграммы взаимодействия и пр.); наиболее известный из них — UML (*Unified Modeling Language*)[70]. Эти *языки отношений* прямо не предназначены для имплементации: уровень абстракции в них слишком высок для трансляции в эффективно работающий код (хотя такие трансляторы существуют); их цель — облегчить репрезентацию сложных программных комплексов.

В результате противоречия между объектом и отношением в конце 1980-х образуется их «синтез» в понятии *шаблона проектирования* (также: *паттерн проектирования* — software design pattern), заимствованном из теории архитектуры[71]. Шаблон проектирования — это схема отношений между объектами, которая выступает общим, повторно используемым (reusable) решением какой-нибудь часто встречающейся в процессе разработки задачи (аналогом служат удачные, проверенные временем способы организации архитектурного пространства). Шаблон проектирования как новый этап эволюции абстракций есть *определяющая* рефлексия отношения объектов и их отношений. Шаблоны как общие решения существовали и ранее, но не были формализованы. В любом относительно крупном программном продукте, основанном на ООП, так или иначе используются какие-то шаблоны проектирования (даже если его создатели не подозревают о существовании этого понятия). Сегодня знание шаблонов (классическими считаются 23 типа, при этом изобретение новых типов не прекращается[72]) является стандартной компетенцией программиста-профессионала.

С точки зрения нашей гипотезы, переход от объектов к шаблонам проектирования «одновременен» переходу к неоплатонизму, в рамках которого также подробно разрабатываются системы отношений между Идеями (в основном иерархические). На эту «одновременность», кроме того, указывает введение в таких языках, как Java, Objective-C, Smalltalk, Common Lisp и др., так называемого *высшего типа* (top type) — универсального класса, или объекта, содержащего в себе все возможные объекты в системе. *Высший тип*, очевидно, гомологичен Единому Плотина: все остальные объекты «эманируют» из него.

Один из таких шаблонов проектирования (разработанный ещё до возникновения самого термина) был положен в основу архитектурного реше-

[69] Kay, Alan. Prototypes vs classes was: Re: Sun's HotSpot. URL: http://lists.squeakfoundation.org/pipermail/squeak-dev/1998-October/017019.html

[70] См.: Jacobson, Ivar; Booch, Grady; Rumbaugh, James. The Unified Software Development Process. Boston: Addison Wesley Longman, 1998.

[71] Alexander, Christopher. A Pattern Language: Towns, Buildings, Construction. Oxford, New York: Oxford University Press, 1977. Об отношении между архитектурой и программированием см. текст К. Александера (1996): Gabriel, ibid., p. v-xi.

[72] См.: Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley Professional, 1994.

ния программ, использующих графический пользовательский интерфейс; именно с его разбора начинается первое руководство по software patterns[73]. Изобретение в конце 1970-х гг. этого шаблона, получившего название *Model–View–Controller*, «одновременно» изобретениям уже не античной мысли, а патристической. Введение именно этой абстракции позволит нам дать финальное определение генезиса GUI.

6 Теология кода

Патристика, насколько нам известно, ещё не разу ни включалась в аналитический аппарат философов computer science — вероятно, потому, что этот период в истории мысли относительно малоизвестен и изолирован. Однако, истолковывая развитие абстракций от античной метафизики к христианской теологии как эволюционно связное, мы можем попробовать спроецировать эту траекторию на эволюцию информатики. Одной из причин, почему теория эволюции кода до сих пор не была создана, возможно, коренится как раз в игнорировании исследователями патристического периода (дальше *аналогий* с платонизмом и аристотелизмом они не шли).

Эволюция знает алломорфозы (преобразования под влиянием среды без изменения общей структуры организма) и ароморфозы (скачкообразные повышения общего уровня организации), но не знает разрывов. Событие христианства могло бы стать таким разрывом в истории мысли, если бы не серия идиоадаптаций христианского послания к чуждой ей античной философии на протяжении первых веков первого тысячелетия. В результате этой конвергенции представления неоплатоников о триадическом строении бытия (Единое—Ум—Душа, по Плотину) соединились с рефлексией трёх главных фигур христианской религии — Отцом, Сыном и Духом — и привели к культурном ароморфозу: созданию концепции Троицы (заимствованной неоплатониками у неопифагорейцев), ставшей сердцем христианской доктрины. Учение о Троице (триадология) знаменовало собой следующий этап развития философем. Прочертим пунктирно траекторию этого развития: 0) досократики: *первовещество*, 1) Платон и Аристотель: *Форма*, 2) неоплатоники: *иерархия Форм* и *Форма Форм* внутри *Единого*, 3) триадология: *отношения* внутри Бога как *трансцензуса Единого* и *Формы Форм* (христианский Бог *сверх*-существен, *сверх*-единичен и т. д.).

Если событие христианства и представляло собой разрыв, то скорее культурно-антропологический: незримый Бог впервые в истории культур явился в облике зримого человека (что теология схватывает в термине *боговоплощение*). Иисус стал первым в собственном смысле *медиумом* — посредником, связующим теологию и антропологию, Небо и Землю, потусторонний мир древности и посюсторонний мир современности. Именно такое понимание события христианства привело Гегеля к мысли о *духе* как

[73] Gamma; Helm; Johnson; Vlissides, ibid, p. 4.

логико-исторической *медиации* (опосредствовании)[74], вдохновило М. Маклюэна на создание *теории медиа*[75] и дало право Р. Дебрэ говорить о *медиологии* как о «профанной христологии»[76].

Это отступление в антропологию и философию религии должно подготовить нас к пониманию абстракций в коде, изобретённых на этапе тематизации объектов и отношений. Прежде всего нужно оговориться, что на этой стадии (конец 1980-х) мышление в computer science перестаёт быть в собственном смысле слова алгоритмическим и становится архитектурным. В развитых средствах ООП-разработки программа представлена не как растущий слоями кристалл (хотя на более низком уровне абстракции она продолжает оставаться текстом, прирастающим новыми строчками), а как векторная карта или план сооружения: таковы графические схемы отношений между объектами в визуальном языке UML. Соответственно, акцент в создании программ переносится с «кустарных» способов производства на систематизацию и организацию программистского труда, коммуникацию между разработчиками и т. д. — всё, что сегодня понимают под термином *программная инженерия* (software engineering).

Одной из наиболее ранних и наиболее употребляемых в индустрии схем отношений между объектами является уже упомянутый паттерн проектирования (точнее, *архитектурный* паттерн — поскольку эта схема сама состоит из нескольких шаблонов) под названием *Model–View–Controller* (MVC).

Паттерн был разработан сотрудником лаборатории *Xerox PARC* по имени Трюгве Реенскауг (Trygve Reenskaug) в 1979 г.[77] Мотивом к созданию этой абстракции послужило стремление усовершенствовать устройство программ, использующих GUI. Идея Реенскауга сводилась к следующему: необходимо так распределить задачи между частями программы, чтобы можно было изменять любую из них, не затрагивая другие, но при этом чтобы эти части работали более слаженно. Согласно предложенной им схеме, в любой программе, использующей GUI, должны присутствовать

1) часть, отвечающая за знания (вычисление, хранение, трансляцию информации), — *Модель* (Model) («Это может быть один объект, что довольно неинтересно, или некая структура объектов»);

2) часть, отвечающая за «визуальную репрезентацию модели», — *Представление*, (View): «Представление привязано к своей модели (или части модели) и получает необходимые для презентации данные, задавая модели вопросы. Оно может также обновлять модель, посылая ей соответствующие сообщения»;

---

[74] Валь, Жан. Несчастное сознание в философии Гегеля. СПб: Владимир Даль, 2006. С. 115.

[75] «В Иисусе Христе нет дистанции или разделения между медиумом и посланием: это единственный случай, когда мы можем сказать, что медиум и послание суть одно и то же» (McLuhan, Marshall. The Medium and the Light: Reflections on Religion. Toronto: Stoddart Publishing, 2002. P. 103).

[76] Дебрэ, ibid., с. 196.

[77] Reenskaug, Trygve. Models – Views – Controllers (10 December 1979). URL: http://heim.ifi.uio.no/trygver/1979/mvc-2/1979-12-MVC.pdf

3) часть, отвечающая за «связь между пользователем и системой»: *Контроллер* (Controller): «Он обеспечивает пользователя вводом, организуя на экране соответствующие представления. Он обеспечивает средства для пользовательского ввода, предоставляя пользователю меню или другие средства управления».

В этой схеме существенно то, что Представление и Контроллер зависят от Модели, но последняя не зависит от первых двух: «контроллер никогда не должен пополнять представления... И наоборот, представление никогда не должно знать о пользовательском вводе»[78]. Такая конфигурация позволяет разрабатывать Модель независимо от её визуального представления, а также создавать несколько различных Представлений для одной Модели. *Только и только* при наличии этих отношений между объектами архитектурный паттерн может быть отнесён к MVC.

По большому счёту, MVC есть не что иное, как обобщение устройства персонального компьютера: 1) память–процессор, 2) экран, 3) клавиатура–мышь. «Главной целью MVC, — писал Реенскауг, — является перекинуть мост между человеческой ментальной моделью и цифровой моделью, существующей в компьютере. Идеальное MVC-решение поддерживает у пользователя иллюзию, что он видит и манипулирует информационной областью напрямую»[79]. Т.е. наряду с чисто техническим мотивом присутствовало также стремление сделать компьютер более доступным в управлении для неспециалиста.

Так, программа с использованием MVC, симулирующая телефонную книгу, будет состоять из трёх объектов: 1) хранимые в некоем внутреннем виде данные, введённые пользователем (имена, номера телефонов и пр.); 2) то, как эти данные отображаются на экране: Представление может меняться, если, например, программа обеспечивает изменения цветовой палитры или сортировку данных по какому-то параметру; 3) различные способы управления телефонной книгой: например, может быть разработан интерфейс для управления при помощи клавиатуры, «мышки» или сенсорного экрана; во всех случаях ни Модель, ни Представление не заботятся об этих вариациях ввода. Таким образом, разработка и модификация сложных программ с графическим интерфейсом становятся более гибкими и застрахованными от глобальных ошибок. Помимо этого, в случае крупных проектов это разделение ответственности (separation of concerns) может стать реальным разделением труда между разными частями программистского коллектива.

Сегодня MVC является стандартом де-факто при разработке GUI. Все его более поздние вариации эксплуатируют эту основную идею[80].

---

[78] Ibid.

[79] Reenskaug, Trygve. MVC. Xerox PARC 1978-79. URL: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

[80] Разработаны, например, такие схемы, как *Hierarchical model–view–controller*, *Model–view–adapter*, *Model–view–presenter*, *Model View ViewModel*, *Presentation–abstraction–control*, *Three-tier architecture* и др.

Таким образом, окончательно уточнённое определение генезиса GUI будет следующим:

(3) $\mathbf{GUI}^{\mathrm{III}} = (\mathbf{клавиатура} + \mathbf{дисплей} + \mathbf{«мышь»})^{\mathrm{I}} + (\mathbf{MVC})^{\mathrm{II}}$

Социокультурное рождение графического пользовательского интерфейса не состоялось бы не только без изобретения новых специальных медиа (графический дисплей, «мышь»), но и без такой абстракции в коде, как MVC, т.е. без метамедиа. Только вместе они составляют GUI как изобретение, а их раздельное рассмотрение само по себе уже является некоей абстракцией (культурологической, социологической...).

Отношение между видимостью (анатомия окна, логика расположения графических элементов и пр.) и стоящими за ней невидимыми абстракциями аналогично в этом случае тому, что было установлено между готической архитектурой и схоластической мыслью[81]: «пользоваться» этой архитектурой можно и без знания о её философской архитектонике, однако невозможно будет понять, почему способы её использования именно таковы (а значит, и способы эти также будут неполными).

Эта аналогия (между нашим исследованием и искусствоведческим) усугубляется тем, что предметом мысли программиста, как и предметом мысли теолога, является *невидимое* как таковое. Согласно классику computer science Фредерику Бруксу (1978), незримость есть сущностное, не акцидентное свойство кода:

> Программный продукт невидим и невизуализуем. <...> Реальность программного обеспечения не встраивается естественным образом в пространство. Поэтому у него нет готового геометрического представления подобно тому, как местность представляется картой, кремниевые микросхемы — диаграммами, компьютеры — схемами соединений. Как только мы пытаемся графически представить структуру программы, мы обнаруживаем, что требуется не один, а несколько неориентированных графов, наложенных один на другой.[82]

Т.е. невозможно визуально представить все слои программной абстракции в их взаимодействии друг с другом. Этот незримый характер кода косвенно подтверждается тем, что до сих не создано ни одного полностью визуального языка программирования, который удовлетворял бы все нужды программиста.

Остановимся теперь более подробно на культурно-философском смысле MVC. Для этого сперва суммируем характеристики отношений между объектами в паттерне:

[81] Имеем в виду классическое исследование Э. Панофски: Panofsky, Erwin. Gothic Architecture and Scholasticism. Latrobe, Pa.: Archabbey Press, 1951.

[82] Брукс, Фредерик. Мифический человеко-месяц, или Как создаются програмные системы. СПб: Символ-Плюс, 1999. С. 169.

1. 1^{o}, 2^{o} и 3^{o} различны, но едины.[83]

2. 2^{o} и 3^{o} зависят от 1^{o}, но 1^{o} не зависит от 2^{o} и 3^{o}.

3. 2^{o} является представлением 1^{o}.

4. 3^{o} связывает 1^{o} и 2^{o}.

5. 1^{o}, 2^{o} и 3^{o} имеют единую сущность.

Мы утверждаем, что такой системы отношений техника прежних времён не знала. Скажем, панель управления в автомобиле не является «представлением» автомобиля: это представление различных аспектов его работы, но работа ещё не есть информация — информация создаётся на самой панели, т.е. Модель неотличима здесь от Представления. При этом само управление автомобилем может обходиться без этого представления, поскольку инструменты управления – руль, коробка передач – сами эту работу уже как-то (за счёт механизма обратной связи) представляют, т.е. здесь, кроме того, неразличимы также Представление и Контроллер. Более близкий к ЭВМ пример — Жаккардовый ткацкий станок: кажется, что в нём есть Модель (конструкция станка), Представление (тканое изделие как вывод) и Контроллер (перфокарты как ввод), однако Представление неотделимо здесь от Модели (изменить Представление, независимо от Контроллера, в этой системе означает изменить саму Модель), и потому схема MVC также неприменима. Наконец, в предшествующем GUI типе интерфейса — интерфейсе командной строки — Представление и Контроллер однозначно отображают модель (в виде текста), поэтому в таком разделении просто нет необходимости. В графическом интерфейсе вывод и ввод становятся многозначными, что требует разделения ответственности и выделения функции представления.

Тем не менее, эта система отношений уже встречалась в культуре: это схема отношений между *Лицами Троицы*, зафиксированная на первых церковных соборах в Символе Веры и разработанная далее в триадологии Отцов Церкви. Именно через сравнительный анализ отношений внутри этих схем можно показать соответствие между ними, выходящее за пределы поверхностного сходства.

Никейский символ веры, принятый на Первом Никейском соборе в 325 г., утверждал, во-первых, зависимость Сына и Духа от Отца (рождение и исхождение соответственно), что отвечает п. 2 нашей суммы характеристик MVC, во-вторых, единство и различие трёх Лиц (п. 1) и, в-третьих,

[83]Единством здесь может быть названо как единство программы, так и единство объекта, в которые входят эти три объекта (MVC — это объект, состоящий из трёх объектов). Последнее, строго говоря, верно только для языков, в которых имеется высший тип и в которых классы являются объектами, напр., в Smalltalk и Objective-C, и с некоторыми оговорками — для других (Java, C#, JavaScript, Visual Basic, Ruby, Python, Perl, Common Lisp, Scala и пр.).

единосущие трёх Лиц (п. 5). Термин «единосущие» (homoousia, консубстанциальность) стал главным теологическим изобретением Никейского собора. За его основу был взят философский термин *ousia*, «сущность» (так, Идея, по Платону, является ousia вещи). Назначением этого изобретения было не допустить распад фигур Троицы на трёх отдельных богов и удержать парадоксальность равенства *1=3*. Особость этих трёх фигур была схвачена Отцами Церкви в термине «ипостась» (hypostasis), обозначающем один из трёх способов существования единой божественной сущности.

Первая наиболее философски разработанная триадология представлена в труде Августина Аврелия «О Троице» (нач. V в.). С целью проиллюстрировать значение Троицы для человеческого познания Августин ввёл понятие «следов Троицы в творении» (vestigia trinitatis in creatura). Такими «следами», в частности, являются:

— *memoria*, *intelligentia*, *voluntas*: «память, понимание и воля — не суть три жизни, но одна жизнь, не суть три ума, но один ум... они суть не три сущности, но одна сущность»[84];

— *re*s, *visio*, *intentio*: «видимое тело, само видение и соединяющее их внимание»[85];

— *memoria*, *visio*, *volitio*: «... Троица возникает из памяти, внутреннего видения и из воли, которая сочетает обоих»[86], и др.

Современный читатель может увидеть в этих «следах» простые когнитивно-психологические модели. Но нужно напомнить, что именно в августиновской триадологии и рождается современное понятие *воли* (которое ещё не было известно грекам): Августин не описывает, но впервые *конструирует* эти модели[87].

На примере действия Лиц Троицы конкретизируется характер отношений между ними. Перечислим гомологии этих отношений с отношениями внутри MVC: ипостась Отца соответствует тому, «что было сохранено в памяти еще прежде того, как оно было представлено» (п. 2, п. 3), ипостась Сына соответствует тому, «что возникает в представлении, когда оно различается» (п. 3), наконец, ипостась Духа есть действие «воли, соединяющей первые два, и посредством этих двух, а также себя самой как третьей, исполняющей единство...» (п. 4: Контроллер связывает Представление с Моделью)[88]. При этом «мы не можем назвать волю ни как бы детищем видения, ибо она была и прежде видения, ни как бы родительницей»[89] (Контроллер не зависит от Представления, а Модель — от Контроллера). Наконец, Отец–Сын–Дух, как и Модель–Представление–Контроллер, различны по ипостасям, но едины по природе (п. 1), и все три единосущны друг другу и объединяющему их целому (п. 5): сущность MVC и каждой из его частей

[84] Августин Аврелий. О Троице. Краснодар: Глагол, 2004. С. 237.
[85] Ibid., с. 241.
[86] Ibid., с. 244-245.
[87] См.: Arendt, Hanna. The Life of the Mind (Combined 2 Volumes in 1). Vol 2. Mariner Books; 1st Harvest/HBJ Ed edition, 1981. P. 84-110.
[88] Августин, с. 252.
[89] Ibid., с. 249.

— объектная, а MVC и есть *model*, модель (как Троица есть Бог).

Стало быть, **Модель, Представление и Контроллер гомологичны «следам» Отца, Сына и Духа** в августиновской интерпретации. Графическая схема паттерна MVC, данная Реенскаугом[90], изоморфна «Щиту Веры» (Scutum Fidei) — схематическому средневековому изображению Троицы. Здесь уместным будет напомнить, что слово *pattern* этимологически восходит к *pater*, «отец»: мышление паттернами есть «патристика» (от *pater* в значении «Отец Церкви») в информатике. Именно с GUI совершается переход от двоичных моделей к троичным, совпадающий с переходом от мышления в терминах двоицы к мышлению в терминах троицы в начале первого тысячелетия. Разумеется, не все шаблоны проектирования являются троичными, поскольку не все из них, как MVC, репрезентируют работу компьютера как целого; однако показательно, что вся совокупность шаблонов проектирования с точки зрения целеполагания разделяется на три типа: *порождающие* (creational), *структурные* (structural) и *поведенческие* (behavioral)[91], — что примерно соответствует ипостасным функциям.

Едва ли обнаруженное соответствие — простая случайность. Нельзя также утверждать, что концепция MVC была напрямую заимствована из христианского богословия. Не менее трудно, если не невозможно, было бы проследить, в духе Макса Вебера, цепочку социокультурных влияний, приведших к повторению этой теологической схемы в технологии программной разработки. Всё это подталкивает нас к необходимости более серьёзно изучить сущность гомологии, что, возможно, является первоочередной задачей, стоящей сегодня перед философией.

Здесь необходимо ответить на возможные возражения относительно гомологии технических и теологических схем[92]. Нам могут сказать: паттерны не являются общеобязательными к употреблению, — это всего-навсего оптимальное решение, к которому программист волен обращаться или не обращаться, — что отличает их от догматов Церкви и соборных теологем. Но, по большому счёту, разнятся только социальные условия, т.е. нормы и санкции, которыми окружены эти культурные артефакты. Антропологически сущность у них одна: и те, и другие разрабатывались с прагматической целью — дабы предотвратить ожидаемые ошибки, возникающие в связи с той или иной схематизацией практики (любая практика начинается с какого-то устойчивого пункта, который принимается на веру, будь то бог или объект; в зависимости от интерпретации этого исходного пункта возникают разные схемы практики). Так, мотивом для введения понятий «единосущие» и «ипостась» была элиминация распространённых ересей первых веков. Ереси (дословный перевод с греческого — «выбор, направление») могут

[90] Reenskaug, ibid.

[91] Gamma; Helm; Johnson; Vlissides, ibid, p. 10.

[92] В этом рассуждении мы в целом следуем мета-антропологической гипотезе Симондона о дивергенции и ре-конвергенции религиозной и технической фаз (Simondon, Du mode..., ibid., p. 159-178) и об *изоморфизме* священного и технического (неопубликованная статья «Psycho-sociologie de la technicité»).

быть поняты тут как «неэффективные» решения той или иной проблемы истолкования. Следование той или иной еретической схеме ведёт к нежелательному поведению, чья нежелательность состоит, главным образом, в опасности антропологического возврата (например, триадологические ереси II-III вв., известные как монархианство, заключались в отрицании триединой природы Бога в пользу Его единства, что могло способствовать возврату к метафизическому монизму греков). История средневековой Церкви — это история ересей и борьбы с ними, так же как история программной инженерии есть постепенная элиминация спорных технологических решений (пример — классическая статья Э. Дейкстры «Go To Statement Considered Harmful» о вреде использования оператора безусловного перехода). В этом смысле Средневековье как интеллектуальная эпоха *теории практики par excellence* наиболее близко по духу компьютерной эре.

Другое, связанное с первым возражение состояло бы в том, что догматы и теологемы являются результатом «открытия», а не «конструирования». Но в разрабатываемой нами перспективе это различие стирается: церковные соборы могут быть уподоблены инженерным симпозиумам, на которых создаются стандарты для индустрии; в то же время программистские изобретения, как мы пытаемся здесь продемонстрировать, следуют предзаданной диалектической логике: новые абстракции в коде в такой же степени являются изобретёнными конструкциями, в какой открытиями новых ступеней развёртывания (представлений о) сущности компьютера.

Сравнения информационных технологий с религиозными феноменами нередки в IT-сообществе (хотя они, естественно, воспринимаются в полукомическом ключе). Известный IT-специалист П. Грэм пишет:

> Языки программирования — это не просто технология, но то, в чём программисты думают. Они наполовину технология и наполовину религия [half technology and half religion].[93]
>
> Как следствие, сравнение языков программирования принимает форму или религиозных войн, или вузовских учебников, которые сознательно настолько нейтральны, что становятся настоящими работами по антропологии. Люди, которые ценят своё спокойствие или хотят получить постоянный преподавательский контракт, избегают говорить на эту тему. Но вопрос только наполовину религиозный; тут есть что поизучать, особенно если вы хотите разрабатывать новые языки.[94]

В свете выявленной нами гомологии технических и теологических схем можно утверждать, что специалистам по computer science «есть что поизучать» на территории обеих «половин».

[93] Graham, Paul. Hackers & Painters: Big Ideas from the Computer Age. Sebastopol, CA: O'Reilly Media, 2009. P. 179.

[94] Ibid., p. 235.

После всего сказанного должно стать ясно, что информатическая гомология антропологического разрыва, учреждённого христианством, находится не в плоскости метамедиа (кода как внутреннего средства коммуникации человека–машина), а в плоскости медиа (внешних средств коммуникации человека–машина): вспомним о данном Р. Дебрэ определении медиалогии как «профанной христологии». Тогда пришествие Христа гомологически соответствует присоединению к ЭВМ графического дисплея, а выделение Духа как особой ипостаси (учреждение *пневматологии*) — соединению дисплея с развитыми средствами ввода (клавиатура + манипулятор). Процитируем ещё раз Ф. Брукса: «Программный продукт невидим и невизуализуем»[95]. Бытие программы как текста подобно бытию Бога Ветхого завета, который пребывает во мраке, но для которого имеется буква (nomen) и который посылает знамения (omen). Иудаизм как эволюционный предшественник христианства есть религия буквы, а не образа; само слово «программа» («для буквы») намекает на гомологическую связь этого этапа эволюции ЭВМ с иудаистической культурой. Но, поскольку, повторимся, речь идёт о гомологиях медиа, а не метамедиа, то с иудаистической культурой нужно сопоставлять не столько этап развития кода, сколько те средства, которые сделали возможным буквенную репрезентацию двоичного кода, «грамматизацию» цифры (на этапе языка ассемблера[96]). Графический дисплей позволил иконически визуализировать невидимую реальность кода, подобно тому как Иисус иконически «визуализировал» невидимую божественную реальность. Графический пользовательский интерфейс, как мы его определили выше, «одновременен» первым этапам христианской теологии. Раннюю реакцию на GUI сторонников интерфейса командной строки[97] можно сравнить с иконоборчеством VI–IX вв.

7 Заключение

Намеченная связь технологии и теологии — которую мы схватываем в термине *техно*теология (как развитие схоластической *онто-теологии*[98]), — открывает необозримое поле для исследований по философии и компьютерным наукам. Основываясь на гипотезе операто́рной рекапитуляции, можно переосмыслить прошлое и настоящее информатики и попытаться предвосхитить её будущее. По мнению разработчика языков Turbo Pascal и C# Андерса Хейлсберга, «...продолжающейся тенденцией в эволюции языков [программирования] является постоянное возрастание уровня абстрак-

[95] Брукс, ibid.

[96] Отметим важное этимологическое совпадение: *assembler*, «сборщик» означает то же, что *logos* (от греч. legein, «собирать»). *В начале был ассемблер* — как возможность перевода нечеловеческого (состояние триггеров в процессоре) в человеческое посредством *буквы*.

[97] Stephenson, ibid.

[98] См.: Хайдеггер, Мартин. Онто-тео-логическое строение метафизики / Хайдеггер, Мартин. Тождество и различие. М.: Гнозис; Логос, 1997.

ции. <...> Ключевой вызов — это поиск следующего уровня абстракции»[99]. Ключевой вызов, бросаемый технотеологией специалистам по computer science, — наличие «гомологических рядов» в развитии *структурной* и *операционной* мысли. Гипотеза операто́рной рекапитуляции даёт разработчикам *метод* поиска абстракций, а именно — *технотеологический перевод*: *перевод* религиозного в техническое и технического в религиозное посредством изобретения *технотеологических схем* (каковые, в силу виртуального характера программистского труда, фактически неотличимы от технических изобретений в традиционном смысле).

Так, современный этап развития computer science (2000-е — 2010-е гг.), по нашим наблюдениям, «одновременен» схоластике. Если развитие западноевропейской метафизики было движением от рефлексии бытия к рефлексии становления, то развитие информатики есть движение от рефлексии становления, от алгоритма, к рефлексии бытия, к *данным* (этим можно объяснить современный бум так называемых big data и интенсивное развитие data mining). Верхней абстракцией в computer science сегодня являются уже не паттерны и не концептуальные схемы, а так называемые *онтологии* — представление знаний о том или ином регионе бытия при помощи иерархий понятий (для чего используются специальные онтологические языки[100]); онтологии применяются в моделировании бизнес-процессов, экспертных системах, искусственном интеллекте. Мышление через онтологии так же систематизирует и формализует мышление через концептуальные схемы, как схоластика систематизирует и формализует патристику. Стандарт онтологического инжиниринга *ISO 15926* (2009) формально и стилистически напоминает средневековые теологические «суммы». В обсуждении информационных онтологий возрождается интерес к *логическому программированию* (эквивалентный интересу к логике у схоластов) и спор об *универсалиях*[101]. Получила распространение технология *программных агентов* (software agents): будучи производными от понятия объекта, *агенты*, в отличие от последнего, имеют некий аналог *свободы воли* во взаимодействии с пользователем или с другими программами, т.е. можно сказать, что они гомологичны понятию *личности*, зарождающемуся в Средние века. Предпринимаются попытки создания универсального языка программирования (Wolfram Language С. Вольфрама, 2014), имеющие культурный эквивалент в *ars magna* Раймонда Луллия (XIII–XIV вв.). Наконец, как альтернатива GUI всё шире применется ZUI — *масштабируемый интерфейс пользователя* (*Zooming User Interface*): в его основе — идея, гомологичная идее *свёртывания/развёртывания* бытия (complicatio/explicatio), возникающая

[99] Biancuzzi, Federico. Masterminds of Programming: Conversations with the Creators of Major Programming Languages. Sebastopol, CA: O'Reilly Media, 2009. P. 311.

[100] Впервые в: Gruber, Thomas. The role of common ontology in achieving sharable, reusable knowledge bases // Principles of Knowledge Representation and Reasoning. Proceedings of the Second International Conference. J.A. Allen, R. Fikes, E. Sandewell (eds.). Morgan Kaufmann, 1991, p. 601-602.

[101] Smith, Barry; Ceusters, Werner. Ontological realism: A methodology for coordinated evolution of scientific ontologies // Applied Ontology. Nov 15; 5(3-4), 2010. P. 139–188.URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3104413/

в платонизме Шартрской школы (XII в.) и позднее взятая на вооружение Николаем Кузанским (XV в.).

Длительный software crisis, в котором находилась информатика, есть проекция технического застоя, имевшего место в Средние века. Теперь яснее становятся слова Алана Кэя, что «настоящая компьютерная революция ещё не случилась»[102]: таковая, по-видимому, наступит при информатической рекапитуляции научно-технической революции. Современная информатика, как кажется, постепенно исчерпывает ресурс «схоластической» мысли и приближается к историческому разрыву, «одновременному» изобретению субъекта в философии Нового времени. На это предположение наталкивает не только общая тенденция в программной инженерии — дрейф от *ремесла* алгоритмов и *мануфактуры* шаблонов проектирования к *индустрии* онтологий, — но и некоторые современные разработки, например, *когнитивные вычислительные платформы* (cognitive computing platforms)[103], в которых уже не человек представляет себе код, а скорее наоборот, весь человеческий мир (географический, социальный, культурный) оказывается представлен коду как субъекту. В теоретическом плане можно ожидать того же процесса накопления внутренних противоречий, который привёл к выходу из средневековой интеллектуальной ситуации: усиление *номиналистических* тенденций в представлении данных и *волюнтаризм* в трактовке способностей (т.е. средств ввода-вывода и абстракций этих средств). Последний может, в частности, означать технологическую революцию по части манипуляторов (до полного неразличения представления в контроллере, т.е. интеллекта в воле, в оккамовской версии), а также установление более самостоятельных и равных отношений с машиной (связанное с ассимиляцией ценностей хакерских сообществ — возможно, вследствие проводимых сегодня на Западе реформ начального и среднего образования). *Аналогию* этого процесса дал уже Д. Рашкофф, говоря о ликвидации компьютерной безграмотности (инкультурации кода) как о цифровой Реформации[104]. Можно предвидеть Ренессанс программистской культуры (когда культура, в том числе *технологическое искусство*, обратится за вдохновением к *классическим* образцам computer science 1960–1970-х гг.), «осень графического интерфейса» и массовое возвращение к интерфейсу командной строки и скриптовым языкам (в согласии с лютеровским принципом *sola Scriptura*, «только Писание»).

Интересной проблемой для изобретательского воображения является гомологическое повторение идей, которые подвели черту под классической философией. Скажем, кантовский «коперниканский переворот» мог бы означать, что абстракции computer science на определённом этапе будут истолковываться уже не как модели для симуляции реальности, а как конструкторы представленного коду человеческого мира (пропущенного сквозь ис-

[102]Kay, Alan. The Real Computer Revolution Hasn't Happened Yet. 2007. URL: http://www.vpri.org/pdf/m2007007a_revolution.pdf

[103]См., напр.: http://www.research.ibm.com/cognitive-computing/

[104]Rushkoff, Douglas. Program or Be Programmed: Ten Commands for a Digital Age. Berkeley: Soft Skull Press, 2011.

торию и географию как *трансиндивидуальные* априорные формы созерцания), а гегелевская идея *опосредствования* (духа) могла бы найти выражение в первом историчностном метамедиуме, чья мощь заключалась бы в рефлексивном пробегании всех этапов своей эволюции посредством единого диалектического механизма. Ницшевское «Бог умер» было бы тогда разрушением субстанциальности всех компьютерных абстракций, а «воля к власти» — началом изобретения новых способов существования кода (в основном через переосмысление сущности искусства как techne). Наконец, неизбежное футуристическое следствие нашей гипотезы — то, что эволюция кода тоже должна будет рано или поздно завершиться: это произойдёт тогда, когда культурный ресурс кода иссякнет и ему станет *больше нечего повторять*. После *конца кода* (который, по нашим грубым подсчётам, совпадает по времени с так называемой «технологической сингулярностью»[105]) футурология сделается археологией, а археология — футурологией.

Особый смысл в свете гипотезы операто́рной рекапитуляции приобретает коммерциализация GUI компанией *Apple*. Компьютер *Apple Macintosh* был первым широко распространившимся компьютером с графическим интерфейсом и, кроме того, первым компьютером, продававшимся без встроенных языков программирования[106]. В *Macintosh* управление машиной (Контроллер, Дух) было по большому счету редуцировано к обращению с графическим интерфейсом (Сын, Представление): языки программирования оказались вытеснены на периферию внимания — началась «пользовательская» культура. Программирование как исторически первый способ коммуникации человек–машина в своём чистом виде не требует опосредования визуальной репрезентацией (её, скажем, не требовало обращение с перфокартами), тогда как современная «пользовательская» коммуникация подразумевает обязательное соподчинение управления визуальному представлению, а также редукцию богатства текстового ввода к манипуляциям с «мышкой». В теологии это соответствует догмату об исхождении Духа «и от Сына» (*филиокве*), концептуально намеченному уже Августином, принятому Западной Церковью в XI в. и ставшему главным теологическим водоразделом между христианским Западом и Востоком: состояние Духа, полагаемого исходящим не только от Отца, но и от Сына, гомологично тому способу имения дела с компьютером, который получил распространение вместе с GUI. Наиболее ярко эта установка была выражена в технотеологическом принципе WYSIWYG (*What You See Is What You Get*, «что видишь, то получаешь»), впервые реализованном компанией *Xerox PARC* в 1974 г. Иными словами, *цифровое филиокве* было заложено в концепцию MVC изначально (что отчасти объясняет преимущественную ориентацию западного неспециального образования на освоение интерфейса различных приложений[107]).

[105] См.: Kurzweil, Ray. The Singularity is Near. New York: Viking Books, 2005.

[106] Harvey, Brian. Is Programming Obsolete? URL: http://www.cs.berkeley.edu/~bh/obsolete.html

[107] Для сравнения: в традициях советской образовательной культуры, генетически не связанной с догмой филиокве, упор делался именно на изучении программирования, в соответствие с идеями академика А.П. Ершова начала 1980-х гг. о программировании как «второй грамотности»: Ершов, А.П. Программирование — вторая грамотность. 1981.

При разработке приложений с сенсорным интерфейсом (multi-touch), в котором контроллер в буквальном смысле *исходит* от представления, *цифровое филиокве* сделалось из теологумена своего рода догматом.

Это антропотеотехническое рассмотрение наводит на мысль о том, что графический пользовательский интерфейс — как посредник между реальностью кода и социальной реальностью, как собственно *социальная репрезентация* ЭВМ — гомологичен средневековой Церкви как посреднику между Богом и людьми, как социальной организации и «телу Христову» одновременно (именно с GUI началась социализация *персональных* компьютеров — компьютеров, которые представлены человеку как лицу, *persona*, и сами выражают «Лицо»). Проблема интерфейса — *экклезиологическая*: отсюда наша характеристика этой проблемы как *точки входа* (без Церкви как общины не было бы развития христианской мысли). Операционные системы, возникающие параллельно с разработкой дисплеев (в начале 1960-х гг.) и первоначально служившие для упрощения рутинных программистских задач, предстают тогда как гомология первых политических систем Средневековья: *Mac OS* «одновременна» первым теократическим государствам, находящимся в сфере влияния латинской Церкви. Что касается так называемых *метафор интерфейса* (interface metaphors: «окно», «рабочий стол» и пр.), то они суть не что иное, как идеологическая надстройка над технотеологическим базисом. Сегодня, когда веб-сервисы постепенно вытесняются приложениями на мобильных платформах, мы можем наблюдать первую неопреодолимую демаркацию кода в сети, приходящую на смену семантически единому пространству веба, подобно тому как возникали первые границы в Европе XIII-XIV вв. Облачные когнитивные платформы могут стать предпосылкой гомологического переизобретения первых *национальных государств* в информационном пространстве. С ликвидацией компьютерной безграмотности (цифровая Реформация), по нашей логике, должна наступить «осень графического интерфейса». Эта ремарка призвана наметить проект исследований по *политической теологии кода*.

Поскольку GUI есть лишь этап в развитии интерфейса, этап, определённый гомологией филиокве, то сам интерфейс может быть технотеологически понят в буквальном смысле — как бытие *inter-faces*, *между-лицами*: 1) между Отцом как данными (данностью) и Сыном как представлением данных (даром Отца), 2) между Сыном как медиумом (периферия) и Духом как метамедиумом (абстракции в коде), 3) между Отцом как чистым различием бытия и ничто (двоичное исчисление) и Духом как мыслью ввиду этого различия (код). Стало быть, **интерфейс не есть реальность, отграниченная от кода, но определённая *технотеологическая интерпретация* реальности кода как метамедиума**. Повторимся: между программированием и использованием компьютера нет фундаментального различия[108]. Коммуникация с компьютером *только* посредством графического интерфейса — это редуцированная, лишённая трансиндивиду-

URL: http://ershov.iis.nsk.su/russian/second_literacy/article.html

[108]Ceruzzi, ibid., p. 81.

ального измерения памяти коммуникация с компьютером посредством программирования: GUI-ввод намеренно обеднён (что скрадывается роскошеством вывода), а его результаты не сохраняются и не становятся достоянием коллектива и общества, подобно библиотекам подпрограмм. Различие между программированием и использованием имеет технотеологический характер. Современная «пользовательская» культура, искусственно отделяющая — посредством *цифрового филиокве* — программирование от использования, может быть тогда расценена как блокировка духа, современная форма *пневмопатологии* (термин политического мыслителя Э. Фёгелина). Решением этой проблемы мог бы стать технотеологически применённый принцип *симметрии* программирования и использования, теологически выраженный Григорием Богословом (IV в.) в формуле *настолько—насколько*[109].

Эволюция абстракций в языках программирования потому подчиняется законам гегелевской диалектики, что код как метамедиум есть гомология духа и сам дух. Но это дух, который путешествует уже не в *истории* как времени человеческих событий (которая, по Гегелю, завершилась, когда свернулась в идее), а в *географии* как пространстве человеческих вещей. Начало этого путешествия духа в вещах было парадоксальным образом засвидетельствовано Марксом в анализе товарной формы. Коль скоро компьютер — это метамедиум, т.е. вещь, способная становиться *всеми* вещами (симулировать любую вещь), *вещь вещей*, то *конец кода* будет означать окончание развёртки духа в вещах — слово станет плотью, наступит *конец географии,* и начнётся подлинно Новое время — время антропотеотехнических гибридизаций[110]. Это настающее время мы по праву могли бы назвать *metamedium aevum*[111].

Теология, обнаруженная в самом сердце кода, — соблазн для богослова и безумие для программиста. Тем не менее, наша работа следует общей тенденции в гуманитарных науках на исследование той остаточной структуры теологического знания, которая скрытым образом присутствует в современной культуре и обществе. Бог как предмет коллективной веры умер, но Его структура оказалась персистентной: теология сохранилась в скелете общества–культуры подобно тому, как оседают в костях тяжёлые металлы. Если наша гипотеза верна, то в XXI в. теология станет для computer science тем же, чем математика стала для естественных наук в XVII в. Вопрос о том, как истолковать это присутствие теологии в коде — как последнюю страницу в *истории* умирания Бога или как Его новое рождение в *вещах*, — мы оставляем открытым.

[109] «Господь воплотился, и стал человеком Бог дольний, чтобы соединиться и стать с ним единым, но и более того, — чтобы и я стал настолько же Богом, насколько Он — человеком» (цит. по: Лурье, Вадим. История Византийской философии. Формативный период. СПб.: Axioma, 2006, с. 77).

[110] О гибридах и переистолковании историографического Нового времени см..: Латур, Бруно. Нового времени не было. Эссе по симметричной антропологии. СПб.: ЕУСПб, 2006. Latour, Bruno. Will Non-Humans be Saved? An Argument on Ecotheology // Journal of the Royal Anthropological Institute (N.S.), №15, 2009, p. 459-475.

[111] Ср.: *medium aevum* — Средневековье.