**Министерство науки и высшего образования**

**Российской Федерации**

**Федеральное государственное бюджетное образовательное**

**учреждение высшего образования**

 **«ТВЕРСКОЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**Математический факультет**

**Кафедра компьютерной безопасности и**

**математических методов управления**

**РЕФЕРАТ**

**по дисциплине “Методы программирования”**

**на тему КОДОГЕНЕРАЦИЯ. ДОКУМЕНТИРОВАНИЕ АРХИТЕКТУРЫ.**

|  |  |  |
| --- | --- | --- |
|  |  | Выполнили:студенты 34 группы математического факультетаспециальностьКомпьютерная безопасностьКокорин Даниил АлександровичМелик-Адамян Вадим РафаэловичПроверила:доцент кафедры КБиММУЦирулева Валентина Михайловна |

**Тверь, 2020 г.**

# Содержание

[Содержание 2](#_Toc41676147)

[Кодогенерация 3](#_Toc41676148)

[Постановка задачи 4](#_Toc41676149)

[Генерация с помощью преобразователя 5](#_Toc41676150)

[Шаблонная генерация 6](#_Toc41676151)

[Документирование архитектуры 8](#_Toc41676152)

[Пользовательская документация программных средств 9](#_Toc41676153)

[Документация по сопровождению программных средств 10](#_Toc41676154)

[Обзор существующих подходов к созданию и документированию архитектуры 11](#_Toc41676155)

[Модель Захмана 12](#_Toc41676156)

[Список источников и литературы 19](#_Toc41676157)

# Кодогенерация

В индустрии программирования наметилась устойчивая тенденция к переходу от объектно-ориентированной технологии программирования к новой парадигме, основанной на теории понятий и языко-ориентированных методах программирования. Причиной этому являются неоправданные затраты для объектно-ориентированной технологии на перевод высокоуровневых понятий и методик предметной области в низкоуровневые конструкции, отвечающие требованиям используемых языков программирования. Этот период разработки программного обеспечения (ПО) является весьма длительным, не особенно творческим, и к тому же практически не нужным в дальнейшем при сопровождении и развитии программного кода – все этапы разработки ПО приходится повторять заново. Альтернативой этому подходу становится новый, использующий предметно-ориентированные языки, которые выполняют роль своеобразного интерфейса, связывая высокоуровневые понятия с непосредственной программной реализацией. Разработанные предметно-ориентированные языки дают возможность сформировать программный код, как непосредственное описание реализованной модели, используя его в дальнейшем для накопления опыта работы и необходимой модификации и развития.

На *рис. 1* изображены два варианты разработки и развития программного обеспечения: первый из них основан на классическом (объектно-ориентированном) подходе, второй – предполагает использование языко-ориентированного программирования, новой парадигмы.

рис. 1. Варианты разработки программного обеспечения: а) классический объектно-ориентированный подход; б) языко-ориентированный подход

Постановка задачи

Сопоставление стадий разработки ПО показывает, что второй вариант обладает существенными преимуществами при моделировании сложных систем, для которых неоднозначны сами модельные представления по описанию процессов в них. Поскольку в этом случае, процесс разработки ПО превращается в непрерывный процесс модификации и развития кода, связанный с необходимостью сравнения различных модельных представлений и выбору оптимальных, а ПО становится в этом случае инструментом, конфигурация которого меняется в зависимости от требований эксперта.

Для обеспечения совместимости по данным компонентов при формировании из них программного приложения, авторами была разработана архитектура компонентной библиотеки, исключающая их непосредственное взаимодействие друг с другом – связь компонентов по данным осуществляется с помощью объектов – посредников. В этом случае, объект, реализующий отдельный алгоритм, взаимодействует только с объектом хранилища данных, получает входные данные из хранилища и возвращает их обратно.

Однако при формировании компонентной библиотеки в рамках разработанной архитектуры появляется дополнительный код в функциональном компоненте (алгоритме). В этом случае алгоритм включает в свой состав не только ядро алгоритма, реализующего непосредственно его функцию, но и оболочку алгоритма, выполняющую связь алгоритма с хранилищем и диском *рис. 2*.

рис. 2. Структура компонента

Объем оболочки (количество строк кода) зависит от объёма данных алгоритма. В среднем одно данное алгоритма требует написания 50 строк программного кода в дополнительных служебных компонентах, реализующих взаимодействие алгоритма с хранилищем.

Автоматизация генерации кода при формировании компонентной библиотеки в такой постановке является нестандартной задачей. Не существует готовых инструментальных средств для автоматизации задач такого класса.

Генерация кода, за исключением самых простых случаев, обычно строится на основе семантической модели. Важной особенностью её применения является разделение семантики и синтаксического анализа. Можно проверить семантику наполнив непосредственно модель и протестировав её, или можно проверить синтаксический анализатор предметно – ориентированного языка, определив, наполняет ли он семантическую модель корректным образом.

Семантическая модель позволяет проверить корректность данных описанных на предметно–ориентированном языке. Например, нет ли алгоритмов, у которых не задан вход или выход, проверить корректность имён данных.

Ниже рассматриваются два основных подхода, используемых при генерации кода: генерация с помощью преобразователя и шаблонная генерация.

Генерация с помощью преобразователя

Генерация с помощью преобразователя основана на написании программы, которая получает на вход семантическую модель, а на выходе даёт исходный текст для целевой языковой среды. Различают преобразователи управляемые вводом и управляемые выводом. Управляемые выводом преобразования начинаются с требуемого вывода и погружаются в семантическую модель для сбора необходимых данных. Преобразование, управляемое входом, проходит по всей семантической модели и генерирует вывод.

На практике часто используются комбинированные преобразователи. В таком варианте преобразователь управляемый выводом используется для разделения выходного документа на логические разделы, а внутренняя часть каждого раздела генерируется на основе конкретных входных данных семантической модели.

Преимуществами рассмотренного метода являются:

1. Простота написания генератора при условии, что генерируется большая часть выходного текста.
2. Возможность реализации многоступенчатого преобразователя, в случаях сложных взаимоотношений между входом и выходом, каждый этап может обрабатывать различные аспекты проблемы.
3. Простое отображение в коде преобразователя обработки сложных случаев при генерации выхода, таких как генерация вывода для элементов коллекции, использующая операторы циклов или условные операторы, когда в зависимости от значения в контексте могут генерироваться различные результаты.

В качестве недостатков выделим следующие:

1. По исходному коду преобразователя сложно понять, как будет выглядеть результат генерации.
2. Структура выходных файлов с исходным кодом жёстко прописана внутри кода преобразователя и обычно разбита на малые части для упрощения преобразователя, что усложняет внесение изменений при изменении формата вывода.
3. При необходимости иметь разные виды выходных файлов на основе одной семантической модели, требуются разные преобразователи.
4. Больший объем кода генератора по сравнению с подходом на основе шаблонов.

Шаблонная генерация

Шаблонная генерация основана на трёх компонентах: обработчик шаблонов, шаблон и контекст. Шаблон представляет собой исходный текст выходного файла, в котором динамические части представлены маркерами-идентификаторами. Последние являются ссылками на контекст, который будет использоваться для заполнения динамических элементов в процессе генерации. Контекст выступает в роли источника динамических данных. Он может быть простой структурой данных или сложной семантической моделью. Разные инструменты используют контексты разных видов. Обработчик шаблонов является инструментом, который связывает шаблон и контекст для генерации выхода.

Управляющая программа будет выполнять шаблонную программу с определенным контекстом и шаблоном для получения выходного файла. Также может использоваться вариант с одним контекстом и разными шаблонами или одним шаблоном и разными контекстами, при этом будут генерироваться разные выходные файлы.

Отметим следующие преимущества, рассматриваемого метода генерации:

1. Взглянув на файл шаблона, можно сразу представить, как будет выглядеть сгенерированный результат.
2. Простота написания обработчика шаблонов, при условии преобладания статического содержимого выходных файлов над динамическими частями.
3. Простота и ясность шаблонов, при условии, что динамическое содержимое достаточно простое.
4. Вынесение описания вывода в отдельные файлы шаблоны позволяет иметь один генератор и набор шаблонов для всех возможных случаев выходных файлов.

В качестве недостатков перечислим следующие:

1. Запутанность шаблонов при сложном динамическом содержимом. Чем больше используется циклов, условий и прочих возможностей языка шаблонов, тем труднее представить, как будет выглядеть результат обработки шаблона.

2. Усложнение и потеря ясности шаблонов при преобладании динамического содержимого над статическим.

3. Неприменимость многоступенчатого подхода в случае сложных взаимоотношений между контекстом и динамическим содержимым шаблонов, только усложнение самого шаблона.

# Документирование архитектуры

При разработке программных средств (далее ПС) создаётся и используется большой объем разнообразной документации. Она необходима как средство передачи информации между разработчиками ПС, как средство управления разработкой ПС и как средство передачи пользователям информации, необходимой для применения и сопровождения ПС. На создание этой документации приходится большая доля стоимости ПС.

Эту документацию можно разбить на две группы:

* + Документы управления разработкой ПС.
	+ Документы, входящие в состав ПС.

*Документы управления разработкой ПС* (software process documentation) управляют и протоколируют процессы разработки и сопровождения ПС, обеспечивая связи внутри коллектива разработчиков ПС и между коллективом разработчиков и менеджерами ПС (software managers) − лицами, управляющими разработкой ПС. Эти документы могут быть следующих типов:

* + *Планы, оценки, расписания*. Эти документы создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения ПС.
	+ *Отчёты об использовании ресурсов в процессе разработки*. Создаются менеджерами.
	+ *Стандарты*. Эти документы предписывают разработчикам, каким принципам, правилам, соглашениям они должны следовать в процессе разработки ПС. Эти стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведётся разработка ПС.
	+ *Рабочие документы*. Это основные технические документы, обеспечивающие связь между разработчиками. Они содержат фиксацию идей и проблем, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов, которые должны войти в ПС.
	+ *Заметки и переписка*. Эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

*Документы, входящие в состав ПС* (software product documentation), описывают программы ПС как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПС).

Эти документы образуют два комплекта с разным назначением:

* + Пользовательская документация ПС (П-документация).
	+ Документация по сопровождению ПС (С-документация).

Пользовательская документация программных средств

Пользовательская документация ПС (user documentation) объясняет пользователям, как они должны действовать, чтобы применить разрабатываемое ПС. Она необходима, если ПС предполагает какое-либо взаимодействие с пользователями. К такой документации относятся документы, которыми должен руководствоваться пользователь при инсталляции ПС (при установке ПС с соответствующей настройкой на среду применения ПС), при применении ПС для решения своих задач и при управлении ПС (например, когда разрабатываемое ПС будет взаимодействовать с другими системами). Эти документы частично затрагивают вопросы сопровождения ПС, но не касаются вопросов, связанных с модификацией программ.

В связи с этим следует различать две категории пользователей ПС: ординарных пользователей ПС и администраторов ПС. Ординарный пользователь ПС (end-user) использует ПС для решения своих задач (в своей предметной области). Это может быть инженер, проектирующий техническое устройство, или кассир, продающий железнодорожные билеты с помощью ПС. Он может и не знать многих деталей работы компьютера или принципов программирования. Администратор ПС (system administrator) управляет использованием ПС ординарными пользователями и осуществляет сопровождение ПС, не связанное с модификацией программ. Например, он может регулировать права доступа к ПС между ординарными пользователями, поддерживать связь с поставщиками ПС или выполнять определенные действия, чтобы поддерживать ПС в рабочем состоянии, если оно включено как часть в другую систему.

Типичный состав пользовательской документации для достаточно больших ПС:

* + Общее функциональное описание ПС. Даёт краткую характеристику функциональных возможностей ПС. Предназначено для пользователей, которые должны решить, насколько необходимо им данное ПС.
	+ Руководство по инсталляции ПС. Предназначено для администраторов ПС. Оно должно детально предписывать, как устанавливать системы в конкретной среде, в частности, должно содержать описание компьютерно-считываемого носителя, на котором поставляется ПС, файлы, представляющие ПС, и требования к минимальной конфигурации аппаратуры.
	+ Инструкция по применению ПС. Предназначена для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для её изучения.
	+ Справочник по применению ПС. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для избирательного поиска отдельных деталей.
	+ Руководство по управлению ПС. Предназначено для администраторов ПС. Оно должно описывать сообщения, генерируемые, когда ПС взаимодействует с другими системами, и как должен реагировать администратор на эти сообщения. Кроме того, если ПС использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

Разработка пользовательской документации начинается сразу после создания внешнего описания. Качество этой документации может существенно определять успех ПС. Она должна быть достаточно проста и удобна для пользователя (в противном случае это ПС, вообще, не стоило создавать). Поэтому, хотя черновые варианты (наброски) пользовательских документов создаются основными разработчиками ПС, к созданию их окончательных вариантов часто привлекаются профессиональные технические писатели. Кроме того, для обеспечения качества пользовательской документации разработан ряд стандартов, в которых предписывается порядок разработки этой документации, формулируются требования к каждому виду пользовательских документов и определяются их структура и содержание.

Документация по сопровождению программных средств

Документация по сопровождению ПС (system documentation) описывает ПС с точки зрения её разработки. Эта документация необходима, если ПС предполагает изучение того, как оно устроена (сконструирована), и модернизацию его программ.

Документация по сопровождению ПС можно разбить на две группы:

1. документация, определяющая строение программ и структур данных ПС и технологию их разработки;
2. документацию, помогающую вносить изменения в ПС.

Документация первой группы содержит итоговые документы каждого технологического этапа разработки ПС. Она включает следующие документы:

* + Внешнее описание ПС (Requirements document).
	+ Описание архитектуры ПС (description of the system architecture), включая внешнюю спецификацию каждой её программы (подсистемы).
	+ Для каждой программы ПС − описание её модульной структуры, включая внешнюю спецификацию каждого включённого в неё модуля.
	+ Для каждого модуля − его спецификация и описание его строения (design description).
	+ Тексты модулей на выбранном языке программирования (program source code listings).
	+ Документы установления достоверности ПС (validation documents), описывающие, как устанавливалась достоверность каждой программы ПС и как информация об установлении достоверности связывалась с требованиями к ПС.

Документация второй группы содержит

* + Руководство по сопровождению ПС (system maintenance guide), которое описывает особенности реализации ПС (в частности, трудности, которые пришлось преодолевать) и как учтены возможности развития ПС в его строении (конструкции). В нем также фиксируются, какие части ПС являются аппаратно- и программно-зависимыми.

Обзор существующих подходов к созданию и документированию архитектуры

На стадии проектирования программных продуктов архитекторы закладывают грандиозные программные замыслы. Но эталонная модель требуемой системы и её фактическая реализация, как правило, сильно различаются.

Трудозатраты группы проектирования можно представить следующим образом:

* 20% трудозатрат – создание модели архитектуры и функциональности программного продукта;
* 70% трудозатрат – воплощение разработанной модели;
* 10% трудозатрат – прочее.

Подобное соотношение затрачиваемых сил демонстрирует, что основной объём ресурсных задач, связанных с реализацией архитектуры и функциональности программных продуктов, концентрируется в области практических процессов, направленных на воплощение спроектированного.

Рабочие компромиссы достигаются за счёт сбалансированного использования доступных ресурсов, технологий и времени. Для достижения заданных целей разработки, проектирования и документирования программных продуктов принято использовать эффективные подходы, которые в состоянии обеспечить требуемый результат. На сегодняшний день выделяют следующие подходы:

* Модель Захмана;
* Модель "4+1";
* Стратегическая модель архитектуры SAM;
* Архитектурные концепции и методики Microsoft;
* Прочие.

Ниже мы рассмотрим один из них.

Модель Захмана

Один из первых и при этом, пожалуй, наиболее распространённый подход к проектированию и описанию архитектуры информационных систем принадлежит Дж. Захману. Предложенная модель (рис. 3) с момента первой публикации эволюционировала с методики описания архитектуры программных продуктов до востребованного метода описания архитектуры компании в целом. Организациями и компаниями, которые применяют её в работе можно считать:

* General motors;
* Bank of America;
* Федеральное правительство США (Архитектура FEAF). FEAF является ответвлением модели Захмана. Эта методика представляет собой самостоятельную модель описания архитектуры, созданную целенаправленно для применения в федеральных и муниципальных органах США. Данный вид модели отличается высокой степенью комплексности политики, процессов и моделей, что отражает специфичные требования к использованию информационных технологий в деятельности американского правительства.
* Методика описания архитектуры Open Group (TOGAF). TOGAF так же, как и FEAF, является самостоятельным ответвлением модели Захмана. Данная методология используется в коммерческих организациях, с развитым уровнем информационных технологий. TOGAF рекомендуется для описания интеграционных компонент, использующихся для поддержки широкого спектра корпоративных приложений, прежде всего, критичных для бизнеса. Этот вид интеграционной архитектуры сильно зависит от принимаемых решений в остальных областях, поэтому в рамках TOGAF рассматриваются смежные домены, влияющие на эффективность архитектуры в целом.
* Методика описания архитектуры министерства обороны США (DoDAF). DoDAF методология построения архитектуры предприятий, созданная по заказу Министерства обороны США (DoD). DoD неоднократно являлось инициатором создания стандартов в области бизнес-моделирования процессов и программных продуктов. Рассматриваемая нами ранее нотация IDEF также создана при непосредственном участии Министерства обороны США. Специфика DoDAF определяется спецификой создания программных продуктов по заказу DoD. DoD – один из крупнейших заказчиков и разработчиков сложных программных продуктов. В предприятиях по созданию таких систем участвуют тысячи подрядчиков, от слаженной работы которых зависит достижение поставленных результатов. Компании, которые стремятся взаимодействовать с DoD на коммерческой основе, должны выполнять DoDAF. Это обстоятельство послужило толчком к началу распространению DoDAF.

рис. 3. Модель Захмана (пример)

Модель Захмана базируется на дисциплине классической, строительной архитектуры, которую мы уже неоднократно упоминали. В первоначальной работе Захман определил архитектуру, как "набор описательных представлений (моделей), которые применимы для описания программного продукта в согласовании с требованиями руководства, которые могут развиваться в течение определенного периода". В соответствии с поставленными целями, была предложена модель архитектуры компании (Zachman Framework for Enterprise Architecture). Модель предлагает решение двух основных архитектурных задач:

* Логически разбить все описание архитектуры на отдельные разделы для упрощения их формирования и восприятия (можно провести параллель с уровнями);
* Обеспечить возможность рассмотрения целостной архитектуры с выделенных точек зрения.

В период создания модели Захмана, в сфере разработки программного обеспечения стала набирать популярность концепция "жизненного цикла", состоящая из следующих стадий:

* Анализ;
* Планирование;
* Проектирование;
* Разработка;
* Тестирование;
* Внедрение;
* Промышленная эксплуатация.

На каждой стадии "жизненного цикла" рассматриваются вопросы, влияющие и определяющие архитектуру и функциональность реализуемого программного продукта. Модернисткой идеей Захмана, изложенной в его модели, было предложение рассмотрения не отдельных частей архитектуры в разные моменты времени, как это подразумевает классическая дисциплина анализа требований, а рассмотрение информационной системы в целом с разных точек зрения/перспектив. Основной посыл этого предположения заключается в возможности обеспечения поэтапного документирования каждого отдельного значимого поведения архитектуры во взаимосвязи со всеми остальными, и создании соответствующего описания.

Чуть ранее мы определились с тем, что когда говорим об архитектуре программных продуктов, то имеем в виду программное обеспечение, которое является сложной, комплексной и высокотехнологичной системой, обладающей относительно большим числом связей и разнонаправленной функциональности, цельное рассмотрение которых в определенный момент времени достаточно затруднительно.

В случае если попытаться рассмотреть отдельную деталь информационного продукта оторвано от системы, в подавляющем количестве случаев – это приводит к спорным и необоснованным решениям касательно настоящего и будущего архитектуры и функционала программного обеспечения. Все детали, которые явно или потенциально взаимосвязаны друг с другом, должны рассматривать едино во взаимосвязи.

Подобная модель рассмотрения архитектуры представлена в модели Захмана, которая по сути отображена в виде таблицы, имеющей 5 строк/уровней и 6 столбцов/колонок.

В строках отражена следующая информация:

* Пара верхних строчек/уровней описывает наиболее общие представления и ожидания от архитектуры программного продукта. На этих уровнях представлена ситуация, которая соответствует интересам менеджеров функциональных процессов, реализуемых в программном продукте;
* Последующий уровень "логической модели" архитектуры, который более определен, по сравнению с верхними соседствующими уровнями, но при этом ещё довольно абстрактен. На этом уровне менеджеры и аналитики, отвечающие за программный продукт, должны работать совместно, коммуницируя на едином языковом пространстве;
* Уровни с 4-ого и далее отражают детали, которые необходимы для участников построения архитектуры. Значимая деталь построения этих уровней заключается в том, что каждый участник, в соответствии со своей ролью, в создании архитектуры и программном продукте, получает/формирует картину в соответствии с необходимым для него уровнем абстракции и детализации;

Колонки определяют и формируют уровни смысловой информации, которая будет составлять базис архитектуры и функциональности программного продукта. Каждая колонка в соответствующем уровне представления состоит из следующего перечня вопросов:

* Что? Используемые данные. Необходимо описать данные, которые будут обоснованно поддерживать реализацию выполнения процессов и функций, на каждом конкретном уровне, в соответствии c заданными показателями качества. От характеристик, целевого назначения и функционального использования данных зависит достижение результата на уровне, в котором они используются. Данные являются необходимым "топливом", приводящим в движение бизнес-процессы организации.
* Как? Процессы и функции. Описываются процессы и функции, специфичные для каждого конкретного уровня. Чем ниже уровень, тем ниже степень неопределённости и абстракции процессов и образующих их функций. Если на верхних уровнях речь идёт об управленческих, то на нижних рассматриваются технические процессы, реализующие и поддерживающие архитектуру и функциональность предприятия или программного продукта.
* Где? Места выполнения этих действий. Отражается информация о том, где должны выполняться конкретные процессы. Если описывается бизнес-процесс, то может быть приведено описание конкретного отдела или географического места, где выполняется значимая функция, в том случае, когда декомпозируется автоматизируемый технический процесс, то уместно привести описание программного продукта или выделенного модуля/компонента, с которым взаимодействуют пользователи системы.
* Кто? Участники - организации и персоналии–участники. Необходимо указать роли и исполнителей, которые выполняют процессы, формирующие архитектуру информационной системы. В том случае, когда выполнение процесса закреплено за организацией, то необходимо указать сферу её ответственности.
* Когда? Управляющие действия. Здесь представлены данные о управляющих действиях, решениях, принятых руководством или закреплённым на законодательном или корпоративном уровне управляющей структуры компании, которые являются движущей силой процессов и обеспечивают их планомерное выполнение и последующее развитие. Здесь могут быть представлены специфичные методологии или политики, которые лежат в основе определенных процессов (тестирования, разработки) конкретной организации.
* Для чего? Цели, задачи и ограничения, определяющие работу системы. Указывается обоснованная мотивация, приведшая к необходимости создания архитектуры. Кроме указания верхнеуровневых целей каждого уровня необходимо отражать задачи, с помощью которых предполагается достигать обозначенные цели и ограничения, формирующие её рамки.

Сам Захман при описании своей модели сравнивал её с таблицей, описывающей повторяющиеся представления компании, и при этом отмечал, что подобное представление необходимо в том случае, когда информационная система становится чем-то большим, чем просто программной процедурой. Сложность и интеграционная комплексность архитектурного проектирования должна реализовываться не неожиданным чудом, а реальным повторяемым процессом.

При заполнении модели таблицы следует соблюдать следующие правила:

* Каждая клеточка таблицы, которая находится на пересечении строки и столбца, должна рассматриваться и заполняться независимо от остальных. Комплексно они образуют функционально полное место для описания базиса программного продукта;
* Каждая клеточка содержит актуальное описание значимых деталей реализации архитектуры информационного продукта в виде определенной модели;
* Базисные модели для каждой из колонок неповторимы.

Модели, каждая из которых включена в определенную ячейку, в совокупности составляют определенный уровень, а уровни образуют описание всей системы.

Каждый столбец представляет собой полное описание системы с определенной перспективы.

При подобных правилах заполнения становится очевидным, что попытка пропуска ячейки, при заполнении любого уровня или столбца, является недопустимой и может привести к смене или потери фокуса разрабатываемой архитектуры. "Дыры" в отдельных ячейках могут привести к тому, что целостная архитектура модифицируется в швейцарский сыр, на котором возвести прочный базис бизнес-процессов современной организации вряд ли получится.

Игнорирование отдельных ячеек или уровней (переход от концептуальной модели бизнес-процесса к её физической реализации), приведёт к разрыву понимания между заказчиками и исполнителями отдельных процессов или функций, что негативно скажется на результатах процессов разработки. Стремления отдельных участников архитектурных процессов к подобной манере работы не так уж редки. На практике это происходит постоянно и связано, в основном, с тем, что бизнес стэйкхолдеры жаждут получить результат как можно скорее.

В модели Захмана подобное отношение к архитектуре программных продуктов является недопустимым.

Подводя итог обсуждения модели Захмана нужно вывести следующие преимущества:

* Ясность и лёгкость понимания;
* Целостность в отношении компании и определенного программного продукта;
* Универсальность для решения разнообразных технических и бизнес задач;
* Возможность обоснованной работы с отдельными абстракциями и сущностями, без потери фокуса по программному продукту в целом;
* Возможность обоснованного принятия решения на отдельных уровнях и высокоточного планирования соответствующих активностей;
* Независимость от применяемого инструментария.

# Список источников и литературы

1. Александров А.Е., Шильманов В.П. Инструментальные средства разработки и сопровождения программного обеспечения на основе генерации кода // Бизнес-информатика №4(22). - М., 2012.
2. Анашкина Н.В. Технологии и методы программирования (конспект лекций). – М.: «Академия», 2012.
3. Иан Соммервилл. Инженерия программного обеспечения = Software Engineering. — 6-е изд. — М.: «Вильямс», 2002. — С. 642. — ISBN 5-8459-0330-0.
4. Подходы к документированию архитектуры программного обеспечения // НОУ ИНТУИТ URL: https://www.intuit.ru/studies/courses/3509/751/lecture/29038 (дата обращения: 29.05.2020).