УДК 004.7

ЗАДАЧИ ОПТИМИЗАЦИИ ПРИ РАЗРАБОТКЕ WEB-ПРИЛОЖЕНИЙ, В РАМКАХ MODX-КОНЦЕПЦИИ

Гутовский Дмитрий Игоревич[1], Добрынин Владимир Николаевич[2]

[1]Аспирант;
ГБОУ ВО МО «Университет «Дубна»,
141980, Россия, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: mfhnb@mail.ru.

[2]Кандидат технических наук, профессор;
ГБОУ ВО МО «Университет «Дубна»,
141980, Россия, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: arbatsolo@yandex.ru.

В статье рассмотрены возможные способы и сценарии применения классических задач оптимизации при разработке и поддержке WEB-приложений. Показан подробный пример аналога задачи о замене оборудования, перенесённый в плоскость WEB-разработки, с соблюдением MODx-концепции. Также, кратко описаны WEB-аналоги и некоторых других задач оптимизации. Приведены аргументы и даны выводы, объясняющие, почему необходимо учитывать задачи оптимизации при разработке WEB-приложений, а также, почему их полноценное соблюдение возможно лишь в рамках MODx-концепции.

Ключевые слова: *CMS*, *MODx*, оптимизация, *MODx*-концепция, сайт, содержимое, *WEB*-сайт, *WEB*, управление, *API*, *HTML*, *CSS*, *JS*, *PHP*, ТПР.

Для цитирования:
Гутовский Д. И., Добрынин В. Н. Задачи оптимизации при разработке WEB-приложений, в рамках MODx-концепции // Системный анализ в науке и образовании: сетевое научное издание. 2022. № 1. С. 91–98. URL : http://sanse.ru/download/463.

THE OPTIMIZATION-TASKS IN THE DEVELOPMENT OF WEB APPLICATIONS, IN THE MODX CONCEPT

Gutovskiy Dmitriy I.[1], Dobrinin Vladimir N.[2]

[1]PhD student;
Dubna State University,
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
e-mail: mfhnb@mail.ru.

[2]PhD in Engineering sciences, professor;
Dubna State University,
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
e-mail: arbatsolo@yandex.ru.

This article shows the possible ways and scenarios to use the classical optimization-tasks for developing and supporting of the WEB-applications. The detailed example of changing equipment task, which is transformed to the WEB-developing area (through the MODx-concept) is shown. Some other WEB-analogs of optimization-tasks are shortly described. There are arguments and conclusions show the necessity to use the optimization tasks in the WEB-developing area but there is shown why it's fully possible only in the MODx-concept.

Keywords: CMS, MODx, optimization, MODx-concept, site, content, WEB-site, WEB, management, API, HTML, CSS, JS, PHP, decision-theory.

For citation:
Gutovskiy D. I., Dobrinin V. N. The optimization-tasks in the development of WEB applications, in the MODx concept. System Analysis in Science and Education, 2022;(1):91–98(In Russ). Available from: http://sanse.ru/download/463.

Введение

WEB-разработка является одним из наиболее перспективных направлений программирования, на сегодняшний день. Это связано с кроссплатформенностью клиентской части *WEB*-приложений. Тот класс программ, который не требует тонкой оптимизации под конкретный тип программного и аппаратного обеспечения, постепенно переходит в *WEB*-пространство, и такие программы охватывают всё более широкий круг потребителей. Вместе с тем, программы, требующие высокой вычислительной эффективности и производительности, а значит и тонкой настройки, постепенно переходят на клиент-серверную архитектуру. Все «тяжёлые» вычисления выполняются на серверах, а на сторону клиента выдаётся *WEB*-интерфейс, позволяющий полноценно работать со всеми возможностями этого ПО. Конечно, есть определённые нюансы, связанные с тем, что не всё ПО можно разделить на клиентскую и серверную часть, например, из-за особенностей сетей, или по соображениям безопасности, однако, общая тенденция в сторону разрастания WEB-архитектуры явно прослеживается [2, 3].

Однако, несмотря на стандартизацию клиентской части *WEB*, с серверной частью не всё так однозначно. Её нельзя полностью стандартизировать, подобно клиентской части, из-за требований высокой вычислительной эффективности. Однако, возможно решить проблему отсутствия стандартизации *API* у *CMS*. Попыткой решения данной проблемы может стать создание стандарта API, универсального для любых *CMS*, на основе *MODx*-концепции. Подробную информацию о данной концепции, а также сравнение классического и *MODx*-подхода к *WEB*-разработке, можно найти в статьях «Определение основных концепций *CMS*» и «Перспективы развития *CMS*».

Сам по себе переход на универсальный *MODx*-стандарт увеличит эффективность разработки *WEB*-приложений и сократит время на их создание, без потери качества. Это связано с полноценной преемственностью модулей для *MODx*-подобных *CMS*, а также, с атомарностью сущностей, из которых состоят модули, написанные под эти *CMS*.

Однако, эффективность разработки WEB-приложений, в случае *MODx*-концепции, можно увеличить ещё больше, применив различные подходы оптимизации, аналогичные тем, что применяются в классических задачах, таких как задача ремонта и замены оборудования, управления ресурсами, размещения производственных объектов и других. Эти задачи (а точнее, их *WEB*-аналоги) возникают и при обычной *WEB*-разработке, однако, их решение не делается в пользу оптимальности, а ограничивается особенностями *API* не *MODx*-подобных систем.

Принципы MODx-концепции

Название *MODx*-концепции взято от имени *CMS*, которая является наиболее популярной и бесплатной, среди систем данной группы. Однако, *MODx* – неединственная *CMS*, подчиняющаяся данной концепции. *MODx*-концепция содержит 9 основных принципов.

Принцип атомарности сущностей выражается в том, что *API CMS* работает с примитивными единицами, которые полностью покрывают все типы данных, необходимые *WEB*-приложению. Однако, среди них нет избыточных типов и нет необходимости в их изменении. Например, поля для ввода текста или для выбора изображения – атомарные сущности, а такие сущности как урок, отзыв о товаре, задание, комментарий и т.д. – неатомарные. *MODx*-подобные *CMS* работают только с атомарными сущностями, а всё остальное собирается из этих примитивов [3].

Принцип атомарности сущностей политики безопасности является частным случаем принципа атомарности сущностей, но выделен в отдельный принцип, так как сущности политики безопасности (разрешения, права доступа, роли) не имеют прямого пересечения с сущностями вывода, которые содержатся в основных модулях сайта.

Принцип преемственности выражается в том, что шаблоны и модули, написанные под определённую версию *CMS*, всегда будут работать с более новой версией этой системы. Конечно, всегда существуют определённые риски. Однако, если речь идёт о *MODx*-подобных системах, то вероятность выхода приложения из строя после обновления *CMS* – минимальна. Данный принцип следует из принципа атомарности сущностей, так как нет нужды в постоянной доработке имеющихся сущностей и добавлении новых, а значит, не нужно постоянно модернизировать *API*.

Принцип независимости адресов файловой системы сайта выражается в отсутствии чёткой привязки элементов *WEB*-приложения к строго определённым адресам на уровне файловой системы сервера, или *WEB*-контекста. Например, во многих стандартных *CMS* регламентируются места расположения шаблонов и модулей, то есть, где и в каких каталогах (с какими путями и названиями) будут находиться модули той или иной группы. В *MODx*-подобных системах таких ограничений нет. Причём, это справедливо, в том числе, и для большинства элементов ядра системы, а не только для шаблонов и дополнений.

Принцип адаптивности выражается в том, что на *MODx*-подобные *CMS* можно добавлять различные модули, несущие разный функционал и написанные под другие (в том числе, и не *MODx*-подобные) системы. При этом, переделка таких «чужеродных» модулей, если и требуется, то минимальная.

Принцип равносильности панели управления и пользовательских страниц выражается в том, что в *MODx*-подобных *CMS* используется один и тот же инструментарий для работы с сущностями панели управления и конечных страниц сайта. То есть, шаблоны и модули панели управления, как и шаблоны пользовательских страниц, создаются с использованием одного и того же инструментария (с одним и тем же *API*). Естественно, можно использовать одни и те же модули для разных контекстов *WEB*-сайта.

Принцип независимости функционала от интерфейса гласит, что серверная логика не должна быть смешана с пользовательскими представлениями (интерфейсом пользователя). То есть, при изменении функционала *WEB*-приложения, интерфейс пользователя не должен меняться, кроме тех случаев, когда это требуется в дизайне, или новый функционал непосредственно затрагивает какой-либо участок интерфейса.

Принцип функциональной расширяемости гласит, что при добавлении нового функционала в любую часть системы, риски некорректной работы существующего – минимальны.

Принцип декомпозиции контекстов позволяет максимально тонко настроить изоляцию различных секций сайта (контекстов). Например, панель управления и пользовательские страницы – это два разных контекста. Контексты можно изолировать по абсолютно разным критериям, например, по правам и ролям пользователей, по группам настроек, с которыми они работают, по языковому разделению, адресам и т.д. При этом, степень изоляции может быть минимальной, и несколько контекстов будут подчиняться одним и тем-же настройкам и правам. Также, может быть и глубокая изоляция, вплоть до того, что будут несколько совершенно разных сайтов, которые, с точки зрения пользователей различных групп (в том числе, и администраторов), не будут связаны между собой. Однако, будет единый центр управления, в котором доступны все контексты (в том числе, которые представляют отдельные сайты). Это позволит создать супер-пользователей, имеющих право централизованного управления. Возможность гибкой регулировки изоляции контекстов позволяет ещё более тонко настраивать любые ограничения и оптимизировать работу сайта в-целом.

В не *MODx*-подобных (то есть, в стандартных) системах не соблюдается основной принцип – принцип атомарности сущностей, что не даёт возможности реализовать остальные принципы. Атомарность сущностей – основополагающая черта *MODx*-подобных систем (и соответствующей концепции).

Применение задач оптимизации в MODx-концепции

При разработке *WEB*-приложений, в рамках *MODx*-концепции, ряд возникающих задач можно представить в виде общепринятых задач, описываемых классическими методами оптимизации. Такие аналогии позволят задействовать более широкий круг специалистов, как при разработке самих конечных приложений, так и при переводе *CMS* в группу *MODx*-подобных. Рассмотрим примеры.

Задача ремонта и замены оборудования. Допустим, что в классическом примере есть набор узлов станка, которые имеют определённые ограничения срока службы и в случае их износа необходимо определить, какое количество элементов (и каких), выгодно заменить, для того, чтобы каждый из элементов отработал максимально возможный для него ресурс, а риск поломки и время простоя оборудования были минимальными. В варианте разработки *WEB*-приложений на основе *MODx-подобной CMS*, в качестве станка можно представить само это приложение, а модули (в том числе и шаблоны страниц), будут узлами станка. Износ подобных элементов связан с устареванием технологий, с помощью которых написан тот или иной модуль, а поломкой может стать, например, открывшаяся уязвимость в плане информационной безопасности модуля, или окончание поддержки технологий, используемых этим модулем (например, языка программирования, на котором он написан). Естественно, как и в случае со станком, важность каждого модуля сайта можно определить, расставив веса и приоритеты обновления этого модуля, а также предусмотреть риски и возможный ущерб, в случае его «поломки» [1, 3]. Конечно, такая задача встаёт при разработке любого *WEB*-приложения, но её решение, вне рамок *MODx*-концепции, невозможно в полной мере. Это связано с особенностями *API* у не *MODx*-подобных систем, из-за которых возникает сильная связность модулей между собой, а это, в свою очередь, накладывает ограничения на обновления. Таким образом, каким бы не был оптимальный план обновления модулей *WEB*-приложения, если *MODx*-концепция в нём не соблюдается, то обновления регламентируются совместимостью модулей с различными версиями *CMS*, а не оптимальностью плана обновления или аварийной необходимостью. К примеру, множество *WEB*-сайтов, на момент 2020-го года, сделаны на базе *CMS Joomla*. Эта система не подчиняется *MODx*-концепции и имеет весьма существенные различия между разными своими версиями. Преемственность модулей сайта, сделанного на основе этой системы, низка. Несмотря на то, что ряд уязвимостей уже давно известен, а их исправления присутствуют только в новых версиях, обновление на них не происходит, так как множество важных компонентов (например, шаблонов сайта), просто не будут работать с новой версией *CMS* (например, при переходе от версии 2 к версии 3). Веса модулей (например, важность тех же шаблонов), можно определить, исходя из того, насколько сложно переписать их под новую версию системы, какие модули будут затронуты побочно, и насколько долго придётся переучивать персонал для работы с новыми модулями, если переписывание старых окажется нецелесообразным.

Например, при задаче замены оборудования, необходимо учесть не только целесообразность замены или ремонта, сравнив, например, стоимость той или иной процедуры, но и готовность персонала к работе с новым оборудованием (если оно существенно изменится). Аналогичная ситуация возникает и в случае с *CMS*. Если *CMS* не подчиняется *MODx*-концепции, то изменение одних модулей часто затрагивает не только логику работы системы, но и интерфейсы пользователя (в том числе, и интерфейс панели управления). Это приводит к необходимости масштабного переобучения персонала, к которому он может быть не готов, да и накладные расходы на такое переобучение могут свести на нет целесообразность замены [1, 3]. Однако, в *MODx*-концепции не возникает проблемы смены интерфейсов, а значит, проблема переобучения персонала решается автоматически. *API MODx*-подобных систем не смешивают серверную логику с пользовательскими представлениями, а значит, при любых манипуляциях и обновлениях, можно вообще ничего не менять для конечного пользователя, что сильно облегчает нахождение оптимального решения вышеописанной задачи [3].

Задача планирования и размещения производственных объектов тоже имеет проекцию на *WEB*-разработку, если речь идёт о *MODx*-концепции. Благодаря принципу независимости адресов файловой системы сайта, различные модули этого сайта могут быть размещены в абсолютно разных местах (как физических, так и программных). Допустим, что при размещении различных объектов какого-либо производства, учитывается множество факторов, начиная от окружающей инфраструктуры, заканчивая транспортным сообщением. Аналогичная задача решается и при условии разработки *WEB*-приложения, если оно является крупномасштабной и сложной системой (а таких, с каждым

годом, становится всё больше). При решении подобной задачи, в контексте разработки сайта, можно более тонко распределять права доступа, настройки резервного копирования, балансировать нагрузку между серверами и так далее. Например, можно выделить пространство для хранения данных определённых групп пользователей на специально-настроенном сервере, где права доступа, процедура авто-резервирования и другие сервисные операции, будут настроены на уровне ОС самого сервера, а не *CMS* сайта. Таким образом, появляются дополнительные рычаги для оптимальной настройки сайта, по средствам размещения его модулей в более подходящем для них окружении. А благодаря межсистемной совместимости модулей у *MODx*-подобных *CMS*, в случае миграции на другую систему из этой группы, изменять настройки конкретного приложения не придётся, так как всё окружение уже настроено и готово к корректной работе. Возникает полная аналогия с вышеприведённой задачей о размещении производственных объектов. Отличия могут заключаться в том, что в классической задаче подобного типа, окружение может быть уже дано, и под него могут подстраивать инфраструктуру, например, фирму по лесозаготовке, рядом с самим лесом. В *WEB*-разработке, окружение могут настроить под объект (модуль) сайта. Однако, могут быть ситуации, в которых происходит постепенное внедрение крупномасштабных систем, и окружение для работы многих модулей уже готово из предыдущих проектов. Гораздо выгоднее внедрить модуль на готовое окружение, чем настраивать всё с нуля (как и в случае с лесозаготовительным предприятием, когда выгоднее построить завод рядом с лесом, а не выращивать лес под конкретный завод, в определённом месте). Соответственно, и в классическом, и в *WEB*-варианте задачи о размещении производственных объектов (в случае с *WEB* – модулей сайта), ситуации могут быть, как с необходимостью подстройки окружения под объект, так и возведении объекта в конкретном месте.

Например, в роли производственного объекта выступает какой-нибудь плагин для анализа статистики, работающий на сайте под управлением *CMS WordPress*. Данная *CMS* не подчиняется *MODx*-концепции, как и подавляющее большинство систем подобного рода (в 2020-ом году). Плагин, описанный выше, должен будет находиться в строго определённом месте сайта, если он полностью соответствует спецификации плагинов для *WordPress*. В противном случае, проблемы дальнейшей поддержки и преемственности возрастут многократно. Конечно, можно произвести ряд настроек на самом сервере, эмулируя виртуальное единое пространство для самой *WordPress*, а на уровне сервера настроить окружение для каждого фрагмента. Однако, это будет не масштабируемо при миграции, да и произвести полноценную индивидуальную настройку всех модулей будет в разы сложнее (и не всегда возможно).

Задача распределения ресурсов тоже может полноценно решаться, при разработке *WEB*-приложений. Однако, это возможно только в тех случаях, когда *CMS* пишется под конкретный проект, или она придерживается *MODx*-концепции. В рамках *MODx*-подобных систем, можно максимально тонко балансировать нагрузку между серверами, например, как описано в случае с файловой системой сайта. Однако, такой баланс может быть не только на уровне ФС, но и при распределении различных ресурсов по контекстам (принцип декомпозиции контекстов). К примеру, различные алгоритмы проверок и шифрования могут применяться для абсолютно любой группы модулей сайта, достаточно просто изолировать её в отдельный контекст. Также, облегчается задача баланса нагрузок между серверной и клиентской частью, благодаря тому, что можно полностью контролировать, какие модули будут вычисляться на стороне сервера, а какие будут переданы на обработку клиентскому устройству.

Плюс к этому, возникает и возможность распределения человеческих ресурсов, так как, благодаря более тонкой настройке прав доступа, нежели в стандартных *CMS*, можно максимально изолировать данные, к которым не должен иметь доступ тот или иной сотрудник, снизив до минимума возможность нелегитимных операций и искусственного повышения прав. Это позволит не возлагать на сотрудников те обязанности, которые нехарактерны для них, исключая излишние звенья проверки. Например, можно легко избежать такой ситуации, когда школьные учителя разных предметов ошибочно заполнили данные, не относящиеся к ним. После этого, системные администраторы (вообще не имеющие отношения к электронным урокам, да они и не должны иметь доступ к ним) исправляют ошибки такого заполнения. Конечно, такие разделения учитываются при разработке любого *WEB*-приложения, в различных инфраструктурах, но именно благодаря тому, что *API MODx*-подобных *CMS* работают с атомарными сущностями, можно максимально тонко разграничить любые роли и права, полностью исключив конфликты неправомерного доступа, при штатной работе системы.

Конечно, это неполное описание *WEB*-интерпретации вышеприведённой задачи, так как модули сайта, не обязательно выражаются файлами, и распределение оных, далеко не всегда происходит на уровне ОС, ФС или *CMS*. Можно привести множество других примеров, в том числе, и распределять модули по различным *CMS*, что позволительно в *MODx*-концепции, однако, это не меняет сути того, что вне рамок данной концепции, полноценная оптимизация размещения модулей сайта не будет возможна, из-за жёстких ограничений *API* у стандартных *CMS*. Таким образом, если и будет предложен оптимальный план размещения модулей, его нельзя будет реализовать (по крайней мере полностью), из-за особенностей *API* и сильной связности модулей в нём.

Для наглядности, рассмотрим подробный пример *WEB*-аналога задачи о замене оборудования. Имеется *WEB*-сайт, с планируемым периодом эксплуатации, равным 5 лет. Прибылью от эксплуатации сайта будем считать средства, сэкономленные в результате снижения затрат различных рабочих ресурсов (в том числе и рабочего времени), благодаря эксплуатации сайта. Как-такового износа сайта, по отношению к эксплуатации, не происходит. Поэтому, прибыль от эксплуатации сайта, на всех этапах, изменяться не будет. Однако, суммарная прибыль будет постепенно снижаться, за-счёт накладных расходов на обслуживание. То есть, как и в случае с оборудованием, расходы на обслуживание будут возрастать, однако, производительность оборудования (то, что названо – прибыль от эксплуатации сайта), в данном случае, не снижается с возрастом. Увеличение затрат на обслуживание связано с усложнением аудита сайта, на предмет ненадёжных узлов, в плане информационной безопасности. Поломка оборудования – прекращение поддержки той технологии, на которой написаны критически-важные модули сайта, либо обнаружение ощутимых проблем информационной безопасности.

Обозначим время, в течение которого используется сайт, за *X*. Прибыль за этап эксплуатации (шаг между принятиями решений) за *R* (допустим, этап = 1 год). Ежегодные затраты на аудит, обозначим за *Z*. Все денежные затраты будут измеряться в ДЕ (денежных единицах).

Табл. 1.Прибыль и затраты

Возраст сайта (*X* лет)	0	1	2	3	4
Годовая прибыль (*R* ДЕ)	98	98	98	98	98
Ежегодные затраты (*Z* ДЕ)	6	12	20	28	36

Решение о сохранении или замене оборудования (в данном случае, под заменой оборудования, будем понимать капитальную переделку ядра сайта), обозначим за *U* (управляющее воздействие). Переменная *U*, в данном случае, будет иметь 2 возможных значения – 0 (сохранить) и 1 (переделать (заменить)). Обозначим суммарную прибыль (с учётом прибыли от эксплуатации сайта и всех затрат на его обслуживание), на каждой итерации (временном этапе), за f_i, а максимальную прибыль на каждой итерации, за F_i. Стоимость кардинальной переделки сайта (эквивалента замены оборудования), примем за *C*. Таким образом, получим следующее: $f_i(X_i\ U_i) = R(X_i) - Z(X_i)$, если $U_i = 0$ (продолжаем эксплуатацию сайта на *i*-том этапе) и $f_i(X_i\ U_i) = R(0) - Z(0) - C$, если $U_i = 1$ (переделываем сайт на *i*-том этапе, следовательно, его возраст (*X*) обнуляется, а затраты на аудит (*Z*) становятся – как у нового).

Запишем уравнение Беллмана: $F_i(X_i) = MAX\{f_i(X_i\ U_i) + F_{i+1}(X_{i+1})\}$. Примем стоимость переделки сайта (*C*) за 32 ДЕ, и решим задачу методом обратной прогонки. Составим таблицу для последнего (5-ого этапа планируемого периода эксплуатации). В 1-ом столбце вписан возраст сайта, во 2-ом расписан случай для продолжения эксплуатации, в 3-ем указан случай замены, в 4-ом выбран наилучший вариант, а в 5-ом указано решение, в соответствии с лучшим вариантом. 1-ым числом (в 1-ом и 2-ом столбцах) указывается прибыль от работы сайта, которая не меняется с возрастом, по условиям задачи. 2-ым числом идут затраты на аудит (которые увеличиваются с возрастом). Во 2-ом столбце, также необходимо вычесть число, равное стоимости переделки сайта (так как там расписан соответствующий случай), однако, во 2-ом столбце не происходит увеличение затрат на аудит, так как каждый раз сайт переделан, а значит, его возраст = 0. В итоге, получаем следующее (см. табл. 2).

Табл. 2. Пятый этап

Возраст	Продолжить (0)	Заменить (1)	Лучший вар.	Итог. реш.
0	98-6=92	98-6-32=60	92	0
1	98-12=86	98-6-32=60	86	0
2	98-20=78	98-6-32=60	78	0
3	98-28=70	98-6-32=60	70	0
4	98-36=62	98-6-32=60	62	0

На следующем этапе, максимальный возраст сайта уменьшается на 1 (так как 1 год планируемого периода эксплуатации уже пройден, и от начала следующего этапа уже не может пройти 5 лет, поэтому, учтено 4 года). Плюс к этому, нужно учесть прибыли, полученные на предыдущих этапах, прибавляя их при расчёте следующих (исходя из уравнения Беллмана, описанного выше). Прибавляемая прибыль не меняется при решении о замене оборудования (переделке сайта), так как затраты на обслуживание (аудит) не растут. Следовательно, условия не ухудшаются, с увеличением возраста. Менялось бы только 1-ое число 3-го столбца, если бы учитывался износ, и прибыль от работы оборудования (сайта) падала с годами, но по условиям задачи, она остаётся неизменной (98 ДЕ). Аналогичная ситуация (со смещением на 1 строку вниз), будет возникать при расчёте следующих этапов.

Табл. 3. Четвертый этап

Возраст	Продолжить (0)	Заменить (1)	Лучший вар.	Итог. реш.
0	98-6+86=178	98-6-32+86=146	178	0
1	98-12+78=164	98-6-32+86=146	164	0
2	98-20+70=148	98-6-32+86=146	148	0
3	98-28+66=136	98-6-32+86=146	146	1

Табл. 4. Третий этап

Возраст	Продолжить (0)	Заменить (1)	Лучший вар.	Итог. реш.
0	98-6+164=256	98-6-32+164=224	256	0
1	98-12+148=234	98-6-32+164=224	234	0
2	98-20+136=214	98-6-32+164=224	224	1

Табл. 5. Второй этап

Возраст	Продолжить (0)	Заменить (1)	Лучший вар.	Итог. реш.
0	98-6+234=326	98-6-32+234=294	326	0
1	98-12+224=310	98-6-32+234=294	310	0

На 1-ом этапе, который начинается с нулевого года эксплуатации, нет необходимости учитывать стратегию замены оборудования (переделки сайта), так как заменять новое на такое же не имеет смысла. Соответственно, не вычитается стоимость замены (32 ДЕ), что даёт следующее выражение: прибыль от работы сайта (98 ДЕ) вычесть затраты на аудит для нового сайта (6 ДЕ) и прибавить максимальную прибыль, просчитанную при анализе предыдущих шагов. Соответственно, получаем следующее: F_0=98-6+310=402 – это максимальная прибыль, а стратегия, ведущая к ней (то есть – лучшая стратегия), описана в таблице ниже (0 – продолжить эксплуатацию, 1 – переделать сайт).

Табл. 6. Итоговая таблица

Возраст	0	1	2	3	4
Решение	0	0	1	0	0

Естественно, что данная задача является сильно упрощённым примером, в котором не учтено множество сопутствующих факторов. Однако, даже из данной задачи видно, что её решение вне рамок *MODx*-концепции, сильно усложняется или становится нецелесообразным. Во-первых, если сайт будет не на *MODx*-подобной *CMS*, значит при его переделке придётся заменить множество модулей и интерфейсов пользователя (принцип преемственности и принцип независимости функционала от интерфейса). Соответственно, прибыль от работы сайта (когда его возраст = 0) будет меньше, чем на следующих этапах эксплуатации (а не одинакова, как в вышеописанном примере), так как придётся проводить переобучение персонала. Плюс к этому, сама переделка сайта, да и изготовление нового (на нулевом этапе эксплуатации), будет значительно сложнее, чем в рамках *MODx*-концепции (принцип атомарности сущностей). Соответственно, возрастёт и стоимость переделки (изготовления) сайта. Зачастую, переделка сайта может быть настолько затратная, а переобучение персонала потребуется весьма глубокое, что переделывать сайт становится нецелесообразным, и эксплуатация продолжается, попытками закрытия уязвимостей своими силами, что снижает разовую стоимость на этапе принятия решения (продолжить дешевле, чем переделать), но постепенно сильно увеличивает затраты на поддержку, ухудшает совместимость с другими модулями и порождает больше рисков потенциальных атак (поломок оборудования).

Заключение

Подводя итог, можно сказать, что в *WEB*-разработке возникают аналоги большинства общепринятых задач, находящихся в области стандартных методов оптимизации. Их решения – тоже могут быть аналогичными. Однако, из-за жёстких ограничений *API* у большинства стандартных *CMS*, такие решения неприменимы на практике. Даже если эти решения будут оптимальны, эффективностью приходится жертвовать, для использования той или иной *CMS*. В *MODx*-концепции подобных проблем не возникает, так как *API* таких систем не имеют жёстких ограничений на разработку любых модулей, а все сущности, которые используются в таких *API* – атомарные.

Несмотря на то, что при разработке на основе *MODx*-подобных систем, появляется возможность решения вышеуказанных задач, это не является обязательным и не усложняет разработку. То есть, если нет острой необходимости в специфической оптимизации каких-либо модулей, то можно обойтись и стандартными настройками, не получив специфических преимуществ оптимизации, но и не потеряв ничего, в сравнении со стандартными *CMS*.

Естественно, описаны – далеко не все классы задач, которые взяты из классических методов оптимизации. Однако, даны примеры тех задач, которые наиболее часто требуют решения, в рамках *WEB*-разработки. При этом, данные оптимизационные задачи не выполняются, из-за отсутствия соблюдения принципов *MODx*-концепции.

Список источников

1. Задача о замене оборудования // Динамическое программирование : лекция. [27 апр. 2020 г. : видеозапись] / Астраханский государственный технический университет. – Время воспроизведения 47:47 – URL: https://www.youtube.com/watch?v=Lh9tFk6jrYI (дата обращения: 17.11.2020)
2. Рael Д. Joomla! : для профессионалов : [усовершенствуйте свой веб-сайт на joomla!] / Дэн Рамел. – Москва [и др.] : Вильямс, 2014. –441 с.
3. Ray B. MODX: The Official Guide. Building dynamic websites with the MODX content management platform / Bob Ray. –Dallas : MODX Press, 2011. –757 с.