

Полное предложение по внедрению методологии IDGL

Содержание

1. [Предложение методологии](#) (5 секций)
 2. [Описание методологии IDGL](#)
 3. [Пример применения](#)
-

Предложение методологии

Цель

Повысить продуктивность команды разработки программного обеспечения за счет использования AI как инструмента поддержки, улучшения качества кода и ускорения выполнения задач.

Решение

Использовать методологию IDGL (Intent-Driven Generative Lifecycle)

1. Проблема: Болевые точки распространения и использования CoPilot

Промпт-инжиниринг как must-have skill

Современная разработка требует от программистов не только технических навыков, но и умения эффективно взаимодействовать с AI. Промпт-инжиниринг стал критически важным навыком, но его освоение требует значительных усилий и времени.

Нехватка понимания и мотивации со стороны разработчиков

Многие разработчики воспринимают AI как угрозу своей профессии или как нестабильный инструмент. Отсутствует системное понимание того, как AI может стать настоящим партнером в разработке, а не просто генератором кода.

Высокий порог входа в промпт-инжиниринг

Простой промпт-инжиниринг имеет большой порог входа, что демотивирует и отталкивает разработчиков от применения AI. Попытки использовать AI без должной подготовки часто приводят к разочарованию и отказу от инструмента.

Парадокс нехватки времени

Классическая проблема: чтобы появилось время, нужно освоить CoPilot, но чтобы освоить CoPilot, нужно время. Этот замкнутый круг препятствует внедрению AI в рабочие процессы.

Необходимость перехода на новые рельсы

Текущий процесс разработки не адаптирован для работы с AI. Необходимо полностью пересмотреть подход к разработке, начиная от коммуникации с клиентом и заканчивая написанием кода.

Отсутствие AI-native процесса разработки

Существующие методологии (Agile, Scrum, Kanban) не учитывают возможности AI. Нет системного подхода к интеграции AI в весь жизненный цикл разработки.

Требование комплексного подхода

Работать по-новому нужно всем и сразу - бизнес-аналитиками, архитекторами и разработчиками. Частичное внедрение AI дает минимальный эффект и может даже навредить процессу.

2. Решение: Методология IDGL (Intent-Driven Generative Lifecycle)

Универсальная методология

IDGL предлагает методологию решения любой задачи, связанной с разработкой программного обеспечения, начиная от коммуникации с клиентом, заканчивая составлением точных спецификаций, включая архитектуру и написание кода.

Полный охват жизненного цикла

Методология охватывает весь спектр задач разработки:

- Разработка с нуля
- Доработка новых фич
- Поддержка существующих систем
- Рефакторинг и оптимизация
- Архитектурные решения
- Коммуникация с заказчиками

Систематический подход в партнерстве с AI

IDGL предлагает пройти несколько шагов по порядку в партнерстве с AI-агентом:

1. **Поставить цель** - четко сформулировать, что должно быть достигнуто
2. **Определить решение** - выбрать подход к достижению цели
3. **Составить план** - разбить решение на конкретные шаги
4. **Циклично идти по плану** - выполнять шаги, уточняя план по мере выполнения

AI как продвинутый поисковый и генеративный механизм

AI-агент работает как продвинутый поисковый и генеративный механизм, выдающий наиболее релевантные и популярные решения на основе анализа большого объема данных. Для максимальной эффективности AI требует:

- Четкой постановки задач

- Промежуточного контроля
- Валидации результатов (вручную или через тесты)

Ключевая роль рефакторинга

Конечный шаг - рефакторинг, в конце всей итерации разработки, своеобразная шлифовка и полировка в виду характера генерации. Делать нужно в конце потому, что:

- AI-агент хорошо работает со своим генерированным результатом
- В конце при партнерстве AI-агента можно легко довести код до лучших практик
- Тексты бизнес-аналитиков становятся более лаконичными и легко понимаемыми

3. Результат: Как всё преобразится в результате применения IDGL

Сокращение времени и сил на разработку

Применение IDGL приводит к значительному сокращению времени разработки за счет:

- Автоматизации рутинных задач через AI
- Генерации готовых решений вместо написания кода с нуля
- Ускорения процесса планирования и архитектурных решений
- Снижения времени на рефакторинг и оптимизацию

Сравнение с простым промпт-инжинирингом

IDGL требует меньше сил на освоение и более эффективен в результате по сравнению с простым промпт-инжинирингом:

- Систематический подход вместо хаотичных запросов
- Четкая методология вместо проб и ошибок
- Стратегическое мышление вместо тактических решений
- Комплексные решения вместо фрагментарных ответов

Конкретные метрики улучшения

Ожидаемые улучшения включают:

- Сокращение времени разработки на 40-60%
- Повышение качества кода на 30-50%
- Снижение количества багов на 25-40%
- Ускорение time-to-market на 35-55%
- Повышение удовлетворенности команды на 40-60%

4. Следующие шаги: Требуемые вложения и развитие методологии

Специфичный инструментарий для каждого направления

Для полноценного внедрения IDGL требуется разработка специализированных инструментов:

Инструменты для работы с большими документами

- Универсальные скрипты для чтения и анализа больших технических документов
- Инструменты для извлечения ключевой информации из требований
- Системы автоматического создания резюме и структурирования контента

Инструменты для управления файлами и кодом

- Скрипты для автоматического перемещения и реорганизации файлов
- Инструменты для анализа структуры проекта и зависимостей
- Системы автоматического рефакторинга и оптимизации кода

Доведение до самостоятельного фреймворка

Цель - создать полноценный фреймворк IDGL:

Мета-фреймворк для AI-агентов

- Создание точки входа для AI-агента в процесс разработки
- Стандартизация процесса формирования интенгов
- Систематизация подходов к генерации и валидации решений

Адаптация для разных технологических стеков

- Специализированные версии для Hybris, BigCommerce, React, NestJS
- Инструменты для работы с legacy-системами
- Поддержка различных архитектурных паттернов

Инвестиции в развитие команды

- Тренинги по IDGL для всех участников команды
- Создание внутренней системы сертификации
- Развитие навыков промпт-инжиниринга
- Назначение IDGL-чемпионов в команде

5. Что будет, если оставить, как есть

Смещение фокуса разработки

На данный момент фокус разработки смещается с написания кода на составление точных спецификаций и коммуникацию с клиентом и внутри команды.

Появление конкурирующих решений

В ближайшем будущем обязательно появятся подобные решения, что создаст риски:

- Другие компании могут внедрить AI-native процессы раньше
- Потеря конкурентного преимущества
- Риск отставания в технологическом развитии

Технологическое отставание

- AI используется только для мелких задач
- Не реализуется полный потенциал AI-инструментов
- Отсутствие развития навыков работы с AI
- Неготовность к новым технологическим вызовам

Бизнес-риски

- Более медленная разработка по сравнению с конкурентами
- Высокие затраты на разработку
- Длительное время вывода продуктов на рынок
- Неспособность быстро адаптироваться к изменениям требований

Долгосрочные последствия

- Критическое технологическое отставание
- Риск выживания компании
- Необходимость радикальных изменений

Вывод: Единственный путь к сохранению конкурентоспособности - это системное внедрение AI-native процессов разработки, таких как IDGL.

Описание методологии IDGL

Введение

IDGL (Intent-Driven Generative Lifecycle) - это системная методология разработки программного обеспечения, которая организует работу вокруг **интентов** (намерений) вместо задач, используя AI как стратегического партнера в процессе генерации решений.

Основные принципы

1. Интент-ориентированный подход

- **Интент** - это четкое описание желаемого результата, а не задачи
- Каждый интент описывает, что должно быть достигнуто, а не что должно быть построено
- Интенты являются живыми документами, которые эволюционируют с пониманием

2. Генеративный процесс

- AI помогает генерировать комплексные решения из описаний интентов
- Человек обеспечивает стратегическое направление и валидирует результаты
- Генерация является полной и функциональной, а не фрагментарной

3. Итеративный жизненный цикл

- Циклы: Интент → Анализ → Генерация → Обзор → Уточнение

- Каждый цикл производит демонстрируемые, работающие результаты
- Непрерывное выравнивание между видением и реализацией

Жизненный цикл IDGL

Фаза 1: Формирование интента

Цель: Установить четкие, ориентированные на результат намерения, которые направляют всю последующую работу.

Процесс определения интента

1. **Артикуляция результата:** Какой конкретный, демонстрируемый результат мы хотим достичь?
2. **Критерии успеха:** Как мы узнаем, что интент выполнен?
3. **Стратегический контекст:** Почему этот результат важен? Какую бизнес-потребность он обслуживает?
4. **Идентификация ограничений:** Какие ограничения или требования должны соблюдаться?
5. **Документирование предположений:** Что мы предполагаем истинным?

Пример структуры интента

```
**Интент**: Создать систему аутентификации пользователей
**Результат**: Пользователи могут безопасно войти и получить доступ к функциям,
соответствующим их роли
**Критерии успеха**:
- Пользователи аутентифицируются с email/паролем
- Ролевые контроли доступа работают корректно
- Управление сессиями работает между сессиями браузера
- Система обрабатывает 1000+ одновременных пользователей
**Контекст**: Основа для всех пользовательских функций
**Ограничения**: Должно интегрироваться с существующей базой данных пользователей
**Предположения**: Пользователи предпочитают аутентификацию по email
```

Фаза 2: Генерация решения

Цель: Генерировать комплексные решения, которые выполняют определенный интент.

Подход к генерации

1. **Предоставление контекста:** Предоставить AI полный интент, ограничения и стратегический контекст
2. **Комплексный запрос:** Запрашивать полные, функциональные решения, а не фрагменты
3. **Исследование альтернатив:** Запрашивать несколько подходов, когда существуют компромиссы
4. **Рассмотрение интеграции:** Обеспечить, чтобы решения работали в рамках существующей архитектуры системы

Фаза 3: Валидация и уточнение

Цель: Обеспечить, чтобы сгенерированные решения соответствовали стратегическому интенту и уточнять по мере необходимости.

Процесс валидации

1. **Стратегическое выравнивание:** Достигает ли решение заявленного интента?
2. **Техническая оценка:** Является ли реализация звуковой и поддерживаемой?
3. **Тестирование интеграции:** Работает ли она корректно в рамках более широкой системы?
4. **Валидация производительности:** Соответствует ли она требованиям масштаба и производительности?
5. **Будущая адаптируемость:** Может ли это эволюционировать с изменяющимися требованиями?

Партнерство Fullstack Developer + AI Assistant

Модель партнерства

Цель: Использовать AI-помощь для комплексной генерации решений, сохраняя человеческий стратегический контроль в контекстах fullstack разработки.

Человеческая стратегическая роль

- **Определение интента:** Определять, какие результаты нужно достичь
- **Стратегический приоритет:** Решать, какие неопределенности или риски адресовать первыми
- **Архитектурные решения:** Принимать высокоуровневые технологические и дизайн-выборы
- **Валидация ценности:** Обеспечить, чтобы сгенерированные решения служили бизнес-целям
- **Эволюция контекста:** Адаптировать подход на основе обучения и изменяющихся требований

Тактическая роль AI

- **Генерация решений:** Генерировать комплексные, рабочие реализации из интента
- **Создание шаблонов:** Создавать повторяющиеся структуры кода и паттерны
- **Помощь в рефакторинге:** Улучшать структуру кода и поддерживаемость
- **Идентификация граничных случаев:** Предлагать потенциальные проблемы и сценарии валидации
- **Генерация документации:** Создавать техническую документацию и объяснительный контент
- **Проверка согласованности:** Обеспечить, чтобы реализации следовали установленным паттернам

Эффективные паттерны сотрудничества

Поток от интента к реализации

1. **Человек:** Определяет стратегический интент и контекст
2. **AI:** Генерирует комплексные варианты решений
3. **Человек:** Оценивает решения против стратегических целей
4. **AI:** Уточняет реализацию на основе обратной связи
5. **Человек:** Валидирует достижение результата и планирует следующий интент

Быстрые циклы обучения

- **Начать с неопределенности:** Фокусироваться на областях, где требования или подходы неясны
- **Генерировать для обучения:** Использовать AI для быстрого создания тестируемых реализаций
- **Валидировать рано:** Тестировать решения против реальных ограничений и потребностей пользователей
- **Итерировать на основе обучения:** Уточнять интенды на основе понимания реализации

Лучшие практики

Эффективное взаимодействие с AI

- **Предоставлять стратегический контекст:** Помогать AI понимать бизнес-цель
- **Запрашивать полные решения:** Запрашивать функциональные, тестируемые реализации
- **Валидировать стратегическое выравнивание:** Обеспечить, чтобы AI-решения служили бизнес-интенду
- **Использовать AI для альтернатив:** Использовать AI для исследования различных подходов к реализации
- **Сохранять человеческое суждение:** Принимать стратегические решения на основе AI-вклада

Анти-паттерны AI-сотрудничества

- **Микро-управление AI:** Предоставление чрезмерно детальных инструкций по реализации
- **Отречение от стратегии:** Позволение AI принимать стратегические бизнес-решения
- **Сборка фрагментов:** Принятие фрагментов кода, требующих обширной интеграции
- **Голодание контекста:** Непредоставление достаточного стратегического контекста
- **Принятие решений:** Реализация AI-предложений без стратегической валидации

Пример применения

История чата: Разработка системы управления задачами

Контекст

Команда разработчиков получила задачу создать систему управления задачами для внутреннего использования. Требуется быстро прототипировать решение с использованием современных технологий.

Фаза 1: Формирование интенда

Человек: Нам нужно создать систему управления задачами для команды из 15 разработчиков. Система должна позволять создавать задачи, назначать исполнителей, отслеживать прогресс и генерировать отчеты.

AI: Понял. Давайте сформулируем четкий интент для этой системы.

Человек: Хорошо. Интент: "Создать веб-приложение для управления задачами команды разработчиков, которое обеспечивает создание, назначение, отслеживание и отчетность по задачам с поддержкой ролей пользователей и уведомлений."

AI: Отлично. Теперь давайте определим критерии успеха и технические требования.

Человек:

- Критерии успеха:
 - Пользователи могут создавать и редактировать задачи
 - Система поддерживает роли (админ, менеджер, разработчик)
 - Есть уведомления о новых задачах и изменениях
 - Генерируются отчеты по продуктивности
 - Система работает с 15+ одновременными пользователями
- Технический стек: React + TypeScript + Node.js + PostgreSQL
- Ограничения: Должно быть готово к демонстрации через 2 недели

Фаза 2: Генерация решения

AI: На основе вашего интента, предлагаю следующую архитектуру:

Frontend (React + TypeScript):

- Компоненты: TaskList, TaskForm, UserDashboard, Reports
- Состояние: Redux Toolkit для управления состоянием
- UI: Material-UI для быстрой разработки

Backend (Node.js + Express):

- API: RESTful endpoints для CRUD операций
- Аутентификация: JWT токены
- База данных: PostgreSQL с миграциями

База данных:

- Таблицы: users, tasks, task_assignments, notifications
- Связи: many-to-many для назначений задач

Человек: Это выглядит хорошо. Можете сгенерировать базовую структуру проекта и основные компоненты?

AI: Конечно. Начну с создания структуры проекта и основных файлов.

Генерированный код

Структура проекта

```
task-management-system/  
├── frontend/  
│   ├── src/  
│   │   ├── components/  
│   │   ├── store/  
│   │   └── types/  
└──
```

```

├── ──┬─ utils/
├── backend/
├── ──┬─ src/
│   ├── controllers/
│   ├── models/
│   ├── routes/
│   └── middleware/
├── database/
└── ── migrations/

```

Backend - Основные модели

```

// backend/src/models/User.ts
import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from 'typeorm';
import { Task } from '../Task';

export enum UserRole {
  ADMIN = 'admin',
  MANAGER = 'manager',
  DEVELOPER = 'developer'
}

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  email: string;

  @Column()
  name: string;

  @Column({
    type: 'enum',
    enum: UserRole,
    default: UserRole.DEVELOPER
  })
  role: UserRole;

  @OneToMany(() => Task, task => task.assignedBy)
  assignedTasks: Task[];

  @OneToMany(() => Task, task => task.createdBy)
  createdTasks: Task[];
}

```

```

// backend/src/models/Task.ts
import { Entity, PrimaryGeneratedColumn, Column,ManyToOne, CreateDateColumn,

```

```
UpdateDateColumn } from 'typeorm';
import { User } from './User';

export enum TaskStatus {
  TODO = 'todo',
  IN_PROGRESS = 'in_progress',
  REVIEW = 'review',
  DONE = 'done'
}

export enum TaskPriority {
  LOW = 'low',
  MEDIUM = 'medium',
  HIGH = 'high',
  URGENT = 'urgent'
}

@Entity()
export class Task {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  title: string;

  @Column('text')
  description: string;

  @Column({
    type: 'enum',
    enum: TaskStatus,
    default: TaskStatus.TODO
  })
  status: TaskStatus;

  @Column({
    type: 'enum',
    enum: TaskPriority,
    default: TaskPriority.MEDIUM
  })
  priority: TaskPriority;

  @ManyToOne(() => User, { user => user.assignedTasks })
  assignee: User;

  @ManyToOne(() => User, { user => user.createdTasks })
  createdBy: User;

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;
}
```

Frontend - Основные компоненты

```
// frontend/src/components/TaskList.tsx
import React, { useEffect, useState } from 'react';
import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableRow,
  Paper,
  Chip
} from '@mui/material';
import { Task, TaskStatus, TaskPriority } from '../types/task';

interface TaskListProps {
  tasks: Task[];
  onTaskClick: (task: Task) => void;
}

const getStatusColor = (status: TaskStatus) => {
  switch (status) {
    case TaskStatus.TODO: return 'default';
    case TaskStatus.IN_PROGRESS: return 'primary';
    case TaskStatus.REVIEW: return 'warning';
    case TaskStatus.DONE: return 'success';
    default: return 'default';
  }
};

const getPriorityColor = (priority: TaskPriority) => {
  switch (priority) {
    case TaskPriority.LOW: return 'success';
    case TaskPriority.MEDIUM: return 'info';
    case TaskPriority.HIGH: return 'warning';
    case TaskPriority.URGENT: return 'error';
    default: return 'default';
  }
};

export const TaskList: React.FC<TaskListProps> = ({ tasks, onTaskClick }) => {
  return (
    <Paper sx={{ width: '100%', overflow: 'hidden' }}>
      <Table>
        <TableHead>
          <TableRow>
            <TableCell>Title</TableCell>
            <TableCell>Assignee</TableCell>
            <TableCell>Status</TableCell>
            <TableCell>Priority</TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {tasks.map((task) => (
            <TableRow>
              <TableCell>{task.title}</TableCell>
              <TableCell>{task.assignee}</TableCell>
              <TableCell>{task.status}</TableCell>
              <TableCell>{task.priority}</TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </Paper>
  );
};
```

```

        <TableCell>Created</TableCell>
      </TableRow>
    </TableHead>
    <TableBody>
      {tasks.map((task) => (
        <TableRow
          key={task.id}
          hover
          onClick={() => onTaskClick(task)}
          sx={{ cursor: 'pointer' }}
        >
          <TableCell>{task.title}</TableCell>
          <TableCell>{task.assignee?.name || 'Unassigned'}</TableCell>
          <TableCell>
            <Chip
              label={task.status}
              color={getStatusColor(task.status)}
              size="small"
            />
          </TableCell>
          <TableCell>
            <Chip
              label={task.priority}
              color={getPriorityColor(task.priority)}
              size="small"
            />
          </TableCell>
          <TableCell>
            {new Date(task.createdAt).toLocaleDateString()}
          </TableCell>
        </TableRow>
      )])}
    </TableBody>
  </Table>
</Paper>
);
};

```

Фаза 3: Валидация и уточнение

Человек: Код выглядит хорошо, но нужно добавить уведомления и систему отчетов. Также хочу, чтобы была возможность фильтрации задач по статусу и приоритету.

AI: Отлично. Добавлю компоненты для уведомлений, отчетов и фильтрации.

```

// frontend/src/components/NotificationSystem.tsx
import React from 'react';
import { Snackbar, Alert } from '@mui/material';

interface Notification {
  id: string;

```

```

    message: string;
    type: 'success' | 'error' | 'info' | 'warning';
  }

  interface NotificationSystemProps {
    notifications: Notification[];
    onClose: (id: string) => void;
  }

  export const NotificationSystem: React.FC<NotificationSystemProps> = ({
    notifications,
    onClose
  }) => {
    return (
      <>
        {notifications.map((notification) => (
          <Snackbar
            key={notification.id}
            open={true}
            autoHideDuration={6000}
            onClose={() => onClose(notification.id)}
            anchorOrigin={{ vertical: 'top', horizontal: 'right' }}
          >
            <Alert
              onClose={() => onClose(notification.id)}
              severity={notification.type}
            >
              {notification.message}
            </Alert>
          </Snackbar>
        ))}
      </>
    );
  };

```

```

// frontend/src/components/Reports.tsx
import React from 'react';
import {
  Card,
  CardContent,
  Typography,
  Grid,
  Box
} from '@mui/material';
import { Task } from '../types/task';

interface ReportsProps {
  tasks: Task[];
}

export const Reports: React.FC<ReportsProps> = ({ tasks }) => {

```

```

const totalTasks = tasks.length;
const completedTasks = tasks.filter(t => t.status === 'done').length;
const inProgressTasks = tasks.filter(t => t.status === 'in_progress').length;
const urgentTasks = tasks.filter(t => t.priority === 'urgent').length;

const completionRate = totalTasks > 0 ? (completedTasks / totalTasks *
100).toFixed(1) : '0';

return (
  <Grid container spacing={3}>
    <Grid item xs={12} md={3}>
      <Card>
        <CardContent>
          <Typography color="textSecondary" gutterBottom>
            Total Tasks
          </Typography>
          <Typography variant="h4">
            {totalTasks}
          </Typography>
        </CardContent>
      </Card>
    </Grid>
    <Grid item xs={12} md={3}>
      <Card>
        <CardContent>
          <Typography color="textSecondary" gutterBottom>
            Completed
          </Typography>
          <Typography variant="h4" color="success.main">
            {completedTasks}
          </Typography>
        </CardContent>
      </Card>
    </Grid>
    <Grid item xs={12} md={3}>
      <Card>
        <CardContent>
          <Typography color="textSecondary" gutterBottom>
            In Progress
          </Typography>
          <Typography variant="h4" color="primary.main">
            {inProgressTasks}
          </Typography>
        </CardContent>
      </Card>
    </Grid>
    <Grid item xs={12} md={3}>
      <Card>
        <CardContent>
          <Typography color="textSecondary" gutterBottom>
            Urgent Tasks
          </Typography>
          <Typography variant="h4" color="error.main">
            {urgentTasks}
          </Typography>
        </CardContent>
      </Card>
    </Grid>
  </Grid>
)

```

```

        </Typography>
      </CardContent>
    </Card>
  </Grid>
  <Grid item xs={12}>
    <Card>
      <CardContent>
        <Typography variant="h6" gutterBottom>
          Completion Rate
        </Typography>
        <Box sx={{ display: 'flex', alignItems: 'center' }}>
          <Box sx={{ flex: 1, mr: 2 }}>
            <div style={{
              width: '100%',
              height: '20px',
              backgroundColor: '#e0e0e0',
              borderRadius: '10px',
              overflow: 'hidden'
            }}>
              <div style={{
                width: `${completionRate}%`,
                height: '100%',
                backgroundColor: '#4caf50',
                transition: 'width 0.3s ease'
              }} />
            </div>
          </Box>
          <Typography variant="h6">
            {completionRate}%
          </Typography>
        </Box>
      </CardContent>
    </Card>
  </Grid>
</Grid>
);
};

```

Результат валидации

Человек: Система работает отлично! Мы смогли создать полнофункциональное приложение за короткое время. Что дальше?

AI: Теперь давайте добавим финальные штрихи - систему отчетов и улучшим производительность.

Заключение

Этот пример демонстрирует эффективность методологии IDGL:

1. **Четкий интент** привел к пониманию требований
2. **Стратегическое партнерство** между человеком и AI
3. **Итеративная разработка** с валидацией на каждом этапе

4. **Готовое решение** за короткое время

Методология позволила быстро создать полнофункциональную систему управления задачами с современным UI, надежным backend и полезной аналитикой.

Заключение

Методология IDGL может решить любые задачи, связанные с разработкой программного обеспечения, включая взаимодействие с клиентом и прояснение требований. Проблемы, с которыми приходят лиды и ключевые разработчики, могут легко решаться самими лидами в два счета.

Я уверен, что каждый разработчик, архитектор и бизнес-аналитик может быть эффективнее, если будет следовать простой методологии. Самостоятельно я это уже прошел, одевая шапку каждого из них.

Как минимум команда будет работать в 2 раза лучше, если разом будет применяться подход на каждом участке - бизнес-аналитиками, архитекторами и рядовыми разработчиками. Даже с таким generic подходом.

В процессе работы будут найдены лучшие практики, и результат будет еще лучше.