Эволюция ПО

Алексей Островский

Физико-технический учебно-научный центр НАН Украины

19 марта 2015 г.

Эволюция ПО

Определение

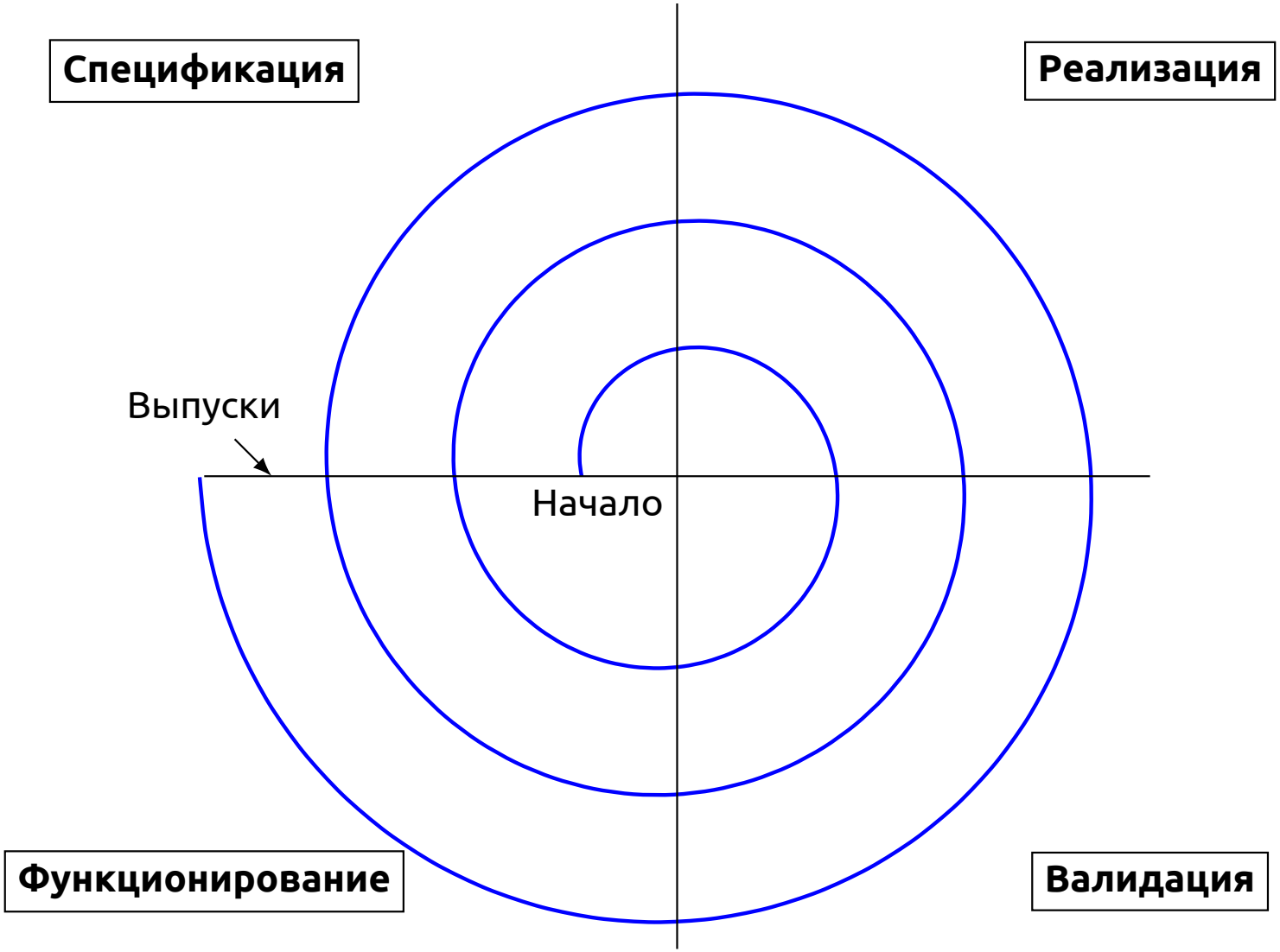
Эволюция ПО (англ. *software evolution*) — процессы разработки, связанные с внесением изменений в программную систему после ее доставки заказчику или конечному пользователю.

В реальных проектах эволюция ПО обычно стоит в $\sim$2 раза больше разработки.

Причины необходимости изменений:

- изменение требований к системе;
- исправление выявленных дефектов;
- изменение среды, в которой выполняется система.

Процесс разработки и эволюции



Спиральный процесс разработки и эволюции ПО

Процессы эволюции ПО

Фазы эволюции ПО

Режимы внесения изменений

Базовый режим: изменения отображаются на все этапы разработки ПО, начиная с формализации в виде требований.

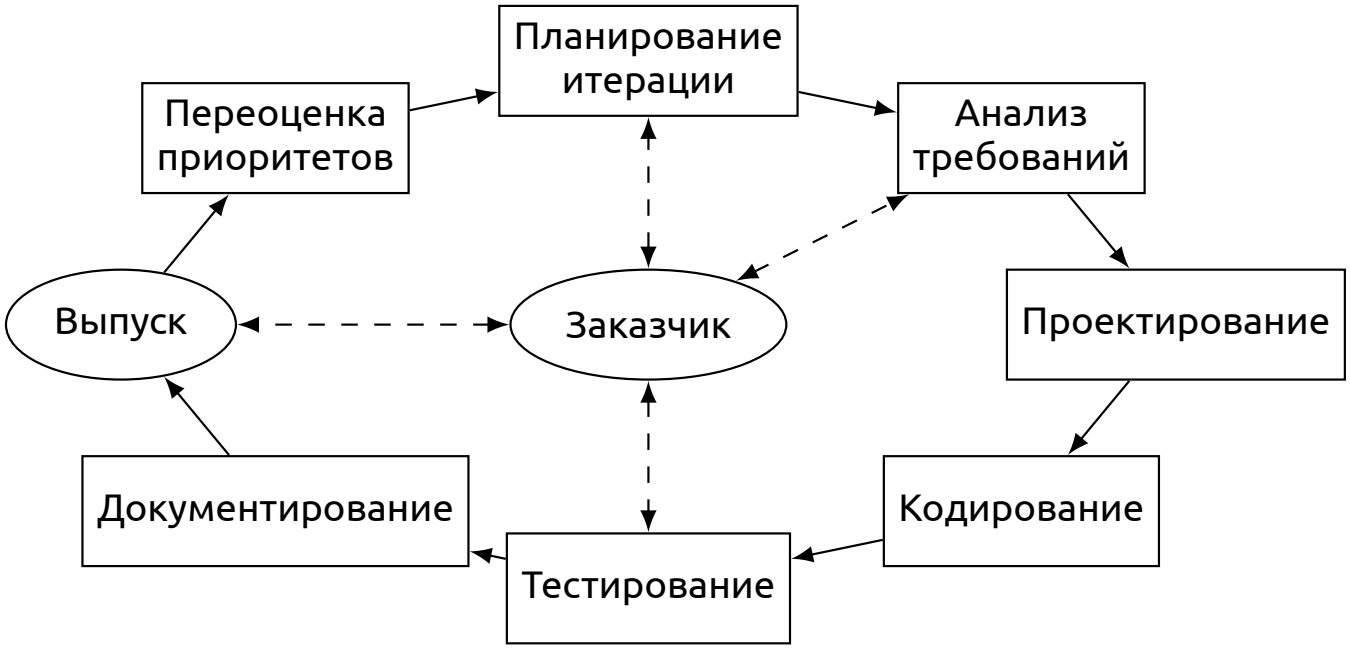
Этапы: запрос на изменение $\longrightarrow$ анализ требований $\longrightarrow$ обновление требований $\longrightarrow$ проектирование $\longrightarrow$ кодирование.

Авральный режим:

- исправление ошибок, мешающих нормальной работе системы;
- изменения окружения ПО, делающих невозможной работу с ней;
- внезапное изменение требований (напр., изменение регулятивных документов).

Этапы: запрос на изменение $\longrightarrow$ анализ исходного кода $\longrightarrow$ правка кода $\longrightarrow$ доставка системы.

Эволюция ПО в гибкой методологии



Эволюция в гибкой методологии — продолжение итераций разработки после доставки ПО.

Инструменты:

- регрессионные тесты (выявляют ошибки при внесении изменений);
- связь с пользователями (идентификация и определение приоритета изменений).

Недостатки: нарушение процесса при разных командах разработки и сопровождения.

Динамика эволюции ПО

Законы эволюции ПО [Lehman, Belady, 1985]:

- **Необходимость изменений** (англ. *continuing change*).
  В программную систему, используемую в реальной среде, необходимо вносить изменения; иначе она становится все менее полезной в своей среде выполнения.
- **Повышение сложности** (англ. *increasing complexity*).
  При эволюции программной системы ее структура в целом усложняется; на поддержание уровня сложности или упрощение структуры ПО нужны дополнительные ресурсы.
- **Эргодичность** (англ. *large program evolution*).
  Эволюция ПО — саморегулирующийся процесс. Характеристики изменений (число ошибок, размер системы, периодичность выпусков) приблизительно одинаковы для всех выпусков.

Динамика эволюции ПО

- **Организационная стабильность** (англ. *organizational stability*).
  Темп разработки программной системы стабилен в течение всего ЖЦ и слабо зависит от затраченных на разработку ресурсов.
- **Сохранение уровня знаний** (англ. *conservation of familiarity*).
  Объем вносимых с каждым выпуском изменений остается стабильным в течение всего периода разработки. (Причина: необходимость сохранения высокого уровня знаний разработчиков о системе.)
- **Цели эволюции** (англ. *continuing growth / declining quality*).
  Для удовлетворения пользователей системе необходимо: **(a)** расширять функциональность; **(b)** адаптировать систему к изменениям в среде выполнения.

Сопровождение ПО

Определение

Сопровождение ПО (англ. *software maintenance*) — организация процессов эволюции программной системы с использованием независимой группы.

Особенности:

- необходимость понимания кода для локализации изменений;
- (потенциально) отсутствие или неполнота документации и спецификации ПО;
- (потенциально) отличная модель жизненного цикла.

Типы сопровождения

- **Исправление дефектов:** дефекты кодирования, проектирования, определения требований (по возрастанию стоимости исправления).
- **Предотвращение дефектов:** устранение скрытых дефектов, которые могут привести к сбоям работы.
- **Адаптация к среде:** внесение модификаций в связи с изменением окружения ПО (оборудования, операционной системы, используемых библиотек, …)
- **Добавление функциональности:** модификация из-за изменения требований по организационным или коммерческим соображениям.

Проблемы добавления функциональности

Наблюдение

Внесение изменений на этапе сопровождение дороже внесения изменений во время основной разработки.

Причины:

- необходимость адаптации к системе и понимания ее кода;
- усложнение процессов сопровождения из-за более «дешевых» решений относительно архитектуры системы на этапе разработки;
- слабая квалификация команды сопровождения, ее незнакомство с предметной областью и / или технологиями, использующимися в системе;
- «устаревание» системы, усложнение понимания и внесения изменений в ее структуру.

Оценка процесса сопровождения

Метрики качества сопровождения:

- число запросов на исправление — при усложнении системы количество вносимых при сопровождении ошибок может превысить число исправляемых дефектов;
- затраты на анализ изменений — оценивает число компонент системы, затрагиваемых запросом на изменение;
- среднее время на реализацию изменения — увеличение подразумевает сильную связь между компонентами системы.

Модели для оценки стоимости сопровождения: COCOMO 2 [Boehm, 2000].

Реинженерия

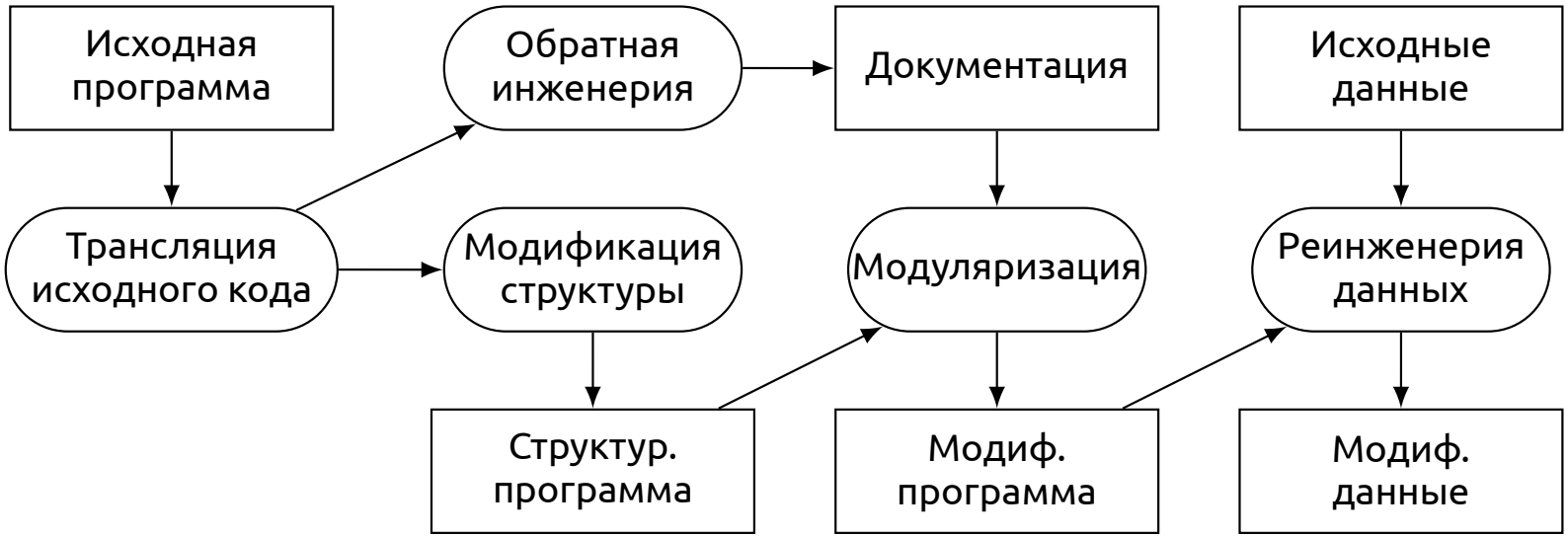
Определение

Реинженерия (англ. *reengineering*) — эволюция программной системы с целью упрощения ее использования, сопровождения или для адаптации к изменившейся среде выполнения.

Преимущества по сравнению с разработкой «с нуля»:

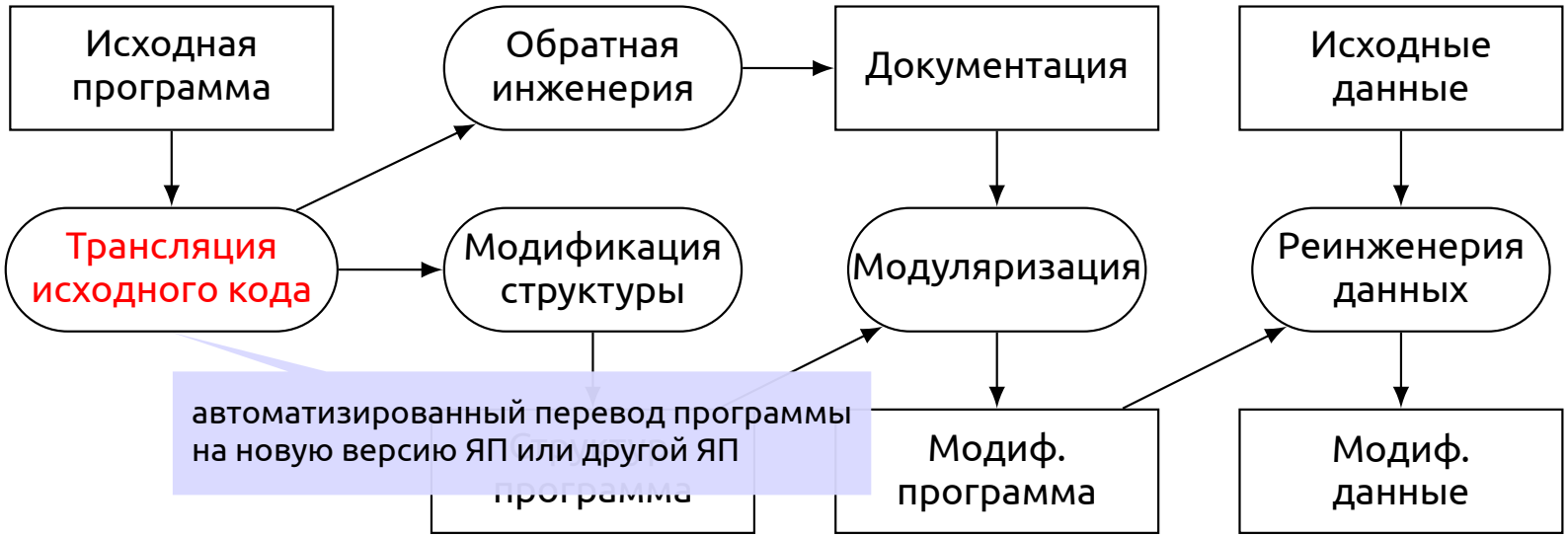
- уменьшение риска;
- сокращение времени разработки;
- уменьшение затрат.

Процессы реинженерии



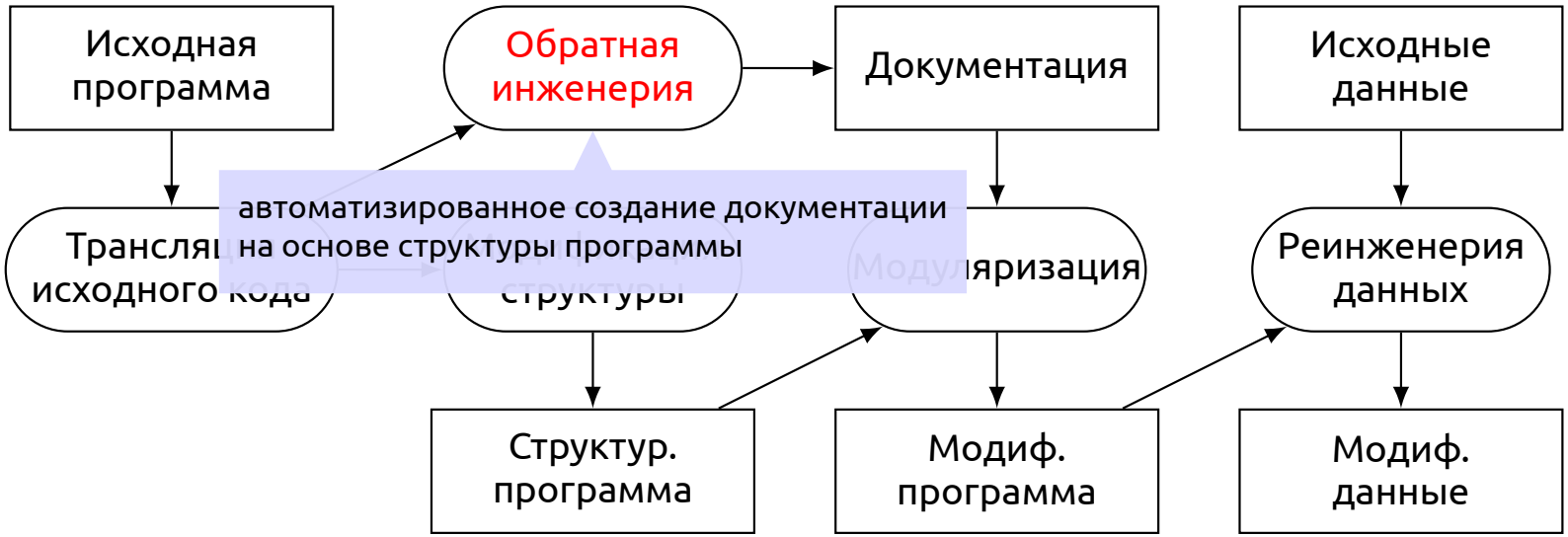
Общая схема процессов реинженерии программной системы

Процессы реинженерии



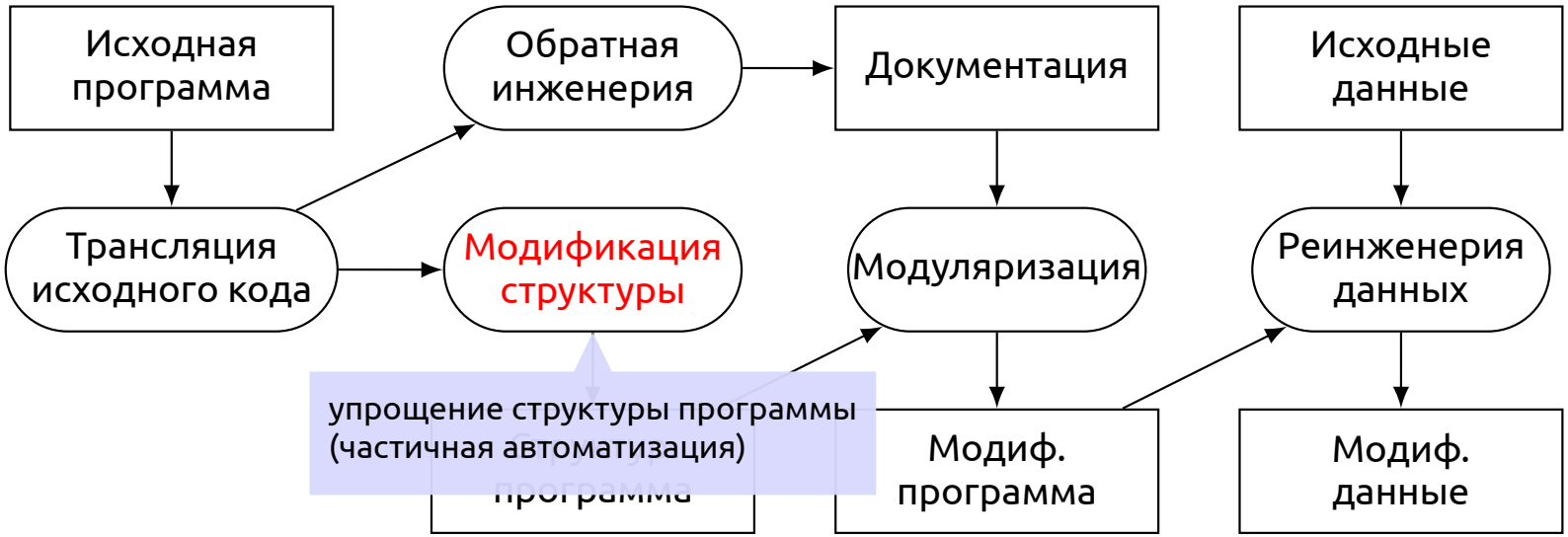
Общая схема процессов реинженерии программной системы

Процессы реинженерии



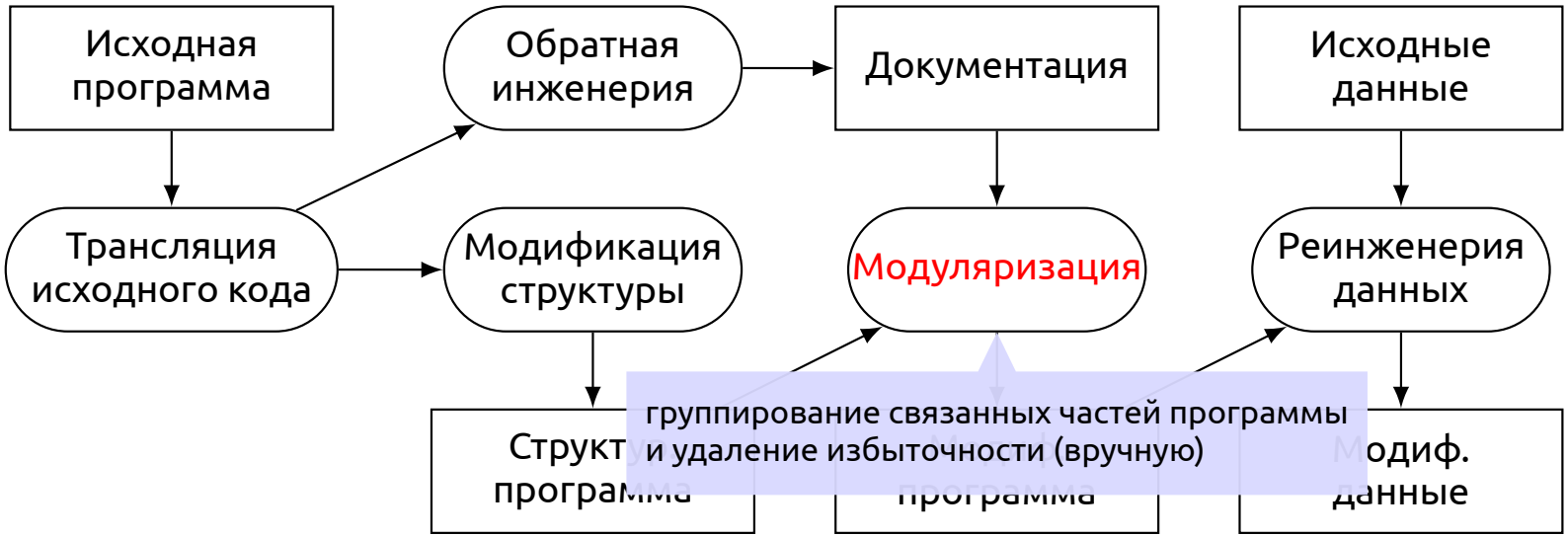
Общая схема процессов реинженерии программной системы

Процессы реинженерии



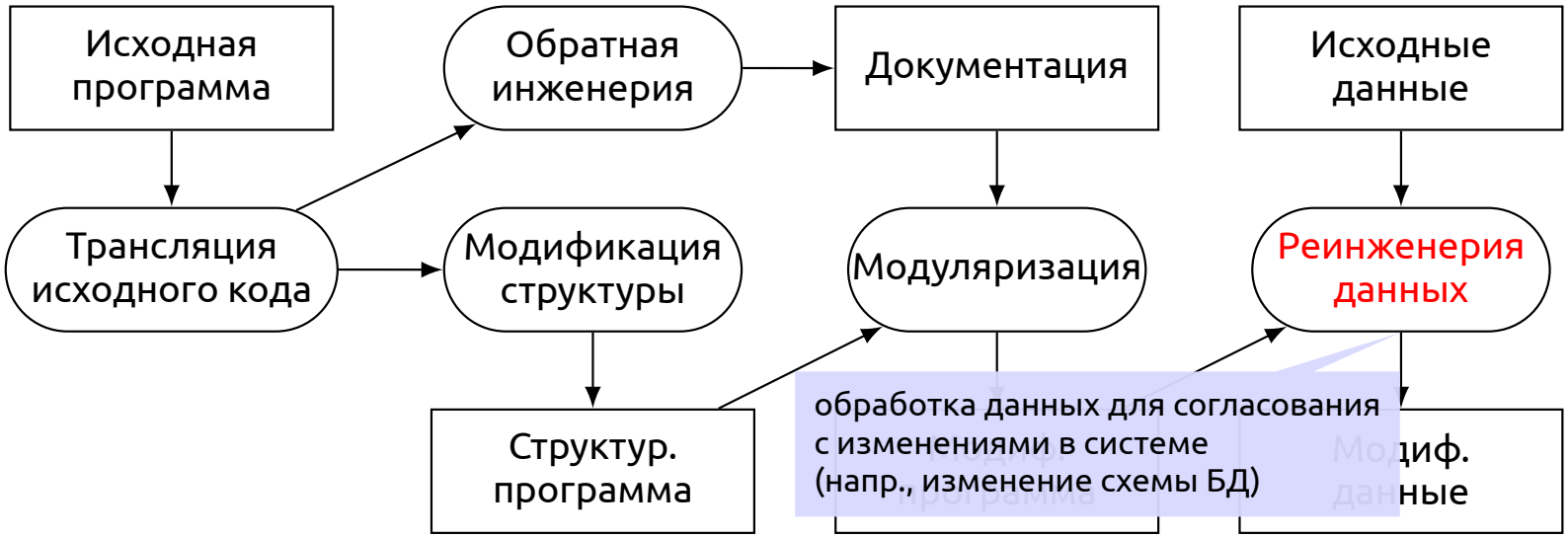
Общая схема процессов реинженерии программной системы

Процессы реинженерии



Общая схема процессов реинженерии программной системы

Процессы реинженерии



Общая схема процессов реинженерии программной системы

Процессы реинженерии

Стоимость процессов реинженерии (по возрастанию):

1. автоматизированное преобразование исходного кода;
2. автоматизированная реструктуризация системы;
3. автоматизированная реструктуризация с дополнительными изменениями;
4. реструктуризация программ и данных;
5. реструктуризация и модификация архитектуры.

Недостатки реинженерии:

- ограниченные возможности инструментов автоматизации;
- высокая стоимость изменения архитектуры ПО;
- более низкое качество сопровождения по сравнению с аналогичной современной системой.

Рефакторинг

Определение

Рефакторинг — процесс усовершенствования программной системы с целью замедлить ухудшение качества ее структуры.

Типы рефакторинга:

- улучшение структуры;
- уменьшение сложности;
- переработка для улучшения понимания.

Отличия от реинженерии:

- применяется как при сопровождении, так и во время разработки;
- должен применяться регулярно;
- меньший масштаб (обычно — отдельные методы и / или поля класса).

Признаки «плохого» кода

- Дублирование сходного / одинакового кода в различных элементах системы.
  Решение: имплементация единого метода или функции с необходимой параметризацией.
- Чрезмерная длина методов.
  Решение: выделение фрагментов кода в более короткие методы.
- Избыточное использование операторов ветвления или конструкций **switch** (**case**).
  Решение: использование полиморфизма.
- Скопление данных — использование одинаковых наборов данных во многих местах программной системы.
  Решение: инкапсуляция данных в виде объекта.
- Чрезмерная универсальность кода.
  Решение: удаление / упрощение избыточного кода.

Примеры рефакторинга

Инкапсуляция поля: осуществление доступа к полю через методы `get*` и `set*`.

До:

```
1  foo.bar = 5;
2  System.out.println(foo.bar);
```

После:

```
1  foo.setBar(5);
2  System.out.println(foo.getBar());
```

Примеры рефакторинга

Обобщение типа: использование наиболее общего возможного типа данных для упрощения повторного использования кода.

До:

```
1  ArrayList<?> list = new ArrayList<?>();
2  public static int min(List<Integer> list);
3  public void examine(Collection<Foo> bar);
```

После:

```
1  List<?> list = new ArrayList<?>();
2  public static int min(Collection<Integer> collection);
3  public void examine(Collection<? extends Foo> bar);
```

Примеры рефакторинга

Pull up / Push down: перенос метода вверх / вниз в иерархии типов.

До:

```
1  public abstract class Figure { /*...*/ }
2  public class Circle extends Figure { /*...*/ }
3  public class Square extends Figure {
4      public rotate(double angle) { /*...*/ }
5  }
```

После:

```
1  public abstract class Figure {
2      public rotate(double angle) { /*...*/ }
3  }
4  public class Circle extends Figure { /*...*/ }
5  public class Square extends Figure { /*...*/ }
```

Примеры рефакторинга

Удаление ветвления: замена ветвления на полиморфизм.

До:

```
1  public class Figure {
2      public static final int SQUARE = 0;
3      public static final int CIRCLE = 1;
4
5      private int type;
6
7      public double getArea() {
8          switch (this.type) {
9              case CIRCLE: /*...*/
10             case SQUARE: /*...*/
11             default: throw new IllegalStateException();
12         }
13     }
14 }
```

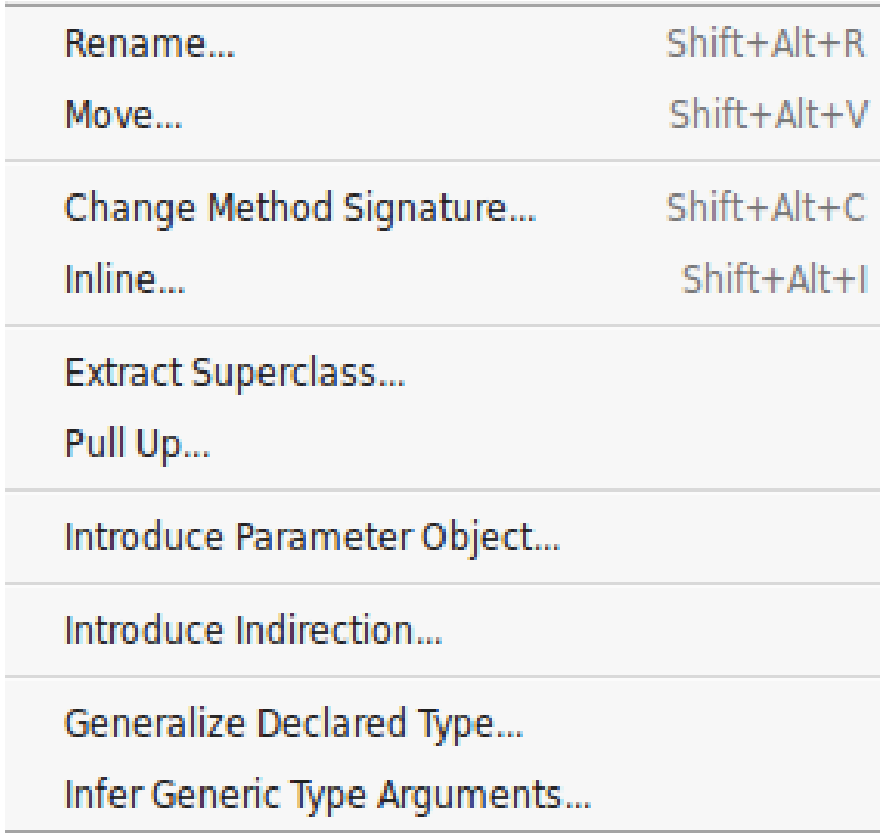
Примеры рефакторинга

Удаление ветвления: замена ветвления на полиморфизм.

После:

```
1  public abstract class Figure {
2      public abstract double getArea();
3  }
4  public class Circle extends Figure {
5      public double getArea() { /*...*/ }
6  }
7  public class Square extends Figure {
8      public double getArea() { /*...*/ }
9  }
```

Автоматизация рефакторинга



Базовые методы рефакторинга реализованы в большинстве современных интегрированных сред разработки, например, Eclipse (изображено контекстное меню этой среды для рефакторинга метода класса).

Работа с устаревшим ПО

Причины использования устаревшего ПО (англ. *legacy software*):

- высокие затраты на разработку нового кода;
- необходимость сертификации.

Сценарии работы с устаревшим ПО:

- сворачивание (в случае устранения зависимостей от системы);
- использование стабильной версии (система необходима, необходимость ее модификации низка);
- реинженерия (качество системы снизилось из-за внесенных изменений; необходима интеграция с новыми компонентами);
- частичная или полная замена ПО (работа с системой невозможна; оправдана разработка новой системы).

Работа с устаревшим ПО

Качество	Значимость	
	Низкая	Высокая
Низкое	сворачивание	реиженерия; замена, если существует готовая походящая система
Высокое	использование стабильной версии; сворачивание, если необходима значительная модификация	использование стабильной версии

Оценка значимости системы

Критерии значимости (определяются заказчиком и конечными пользователями):

- интенсивность и частота использования системы;
- поддерживаемые на текущий момент производственные процессы;
- функциональная надежность системы (англ. *dependability*) — корректность результатов работы системы при условии наличия в ней дефектов;
- важность данных, генерируемых системой.

Оценка качества системы

Критерии качества взаимодействия с окружением:

- стабильность поставщиков системы (ответственных за доставку, разворачивание и сопровождение);
- частота отказов системы и окружения;
- возраст оборудования и ПО, стоимость их сопровождения;
- производительность окружения;
- требования, касающиеся поддержки вспомогательного ПО и оборудования;
- затраты на сопровождение (напр., замену оборудования и продление лицензий на вспомогательное ПО);
- интероперабельность (проблемы взаимодействия с другим ПО, в частности, при сборке системы; потребность в эмуляции оборудования).

Оценка качества системы

Критерии качества самой системы:

- понятность исходного кода и дизайна;
- наличие и полнота документации;
- наличие и согласованность схемы данных;
- производительность и ее влияние на пользователей;
- используемые языки программирования;
- наличие управления конфигурацией и описания версий компонентов системы;
- наличие и полнота тестовых сценариев;
- навыки группы сопровождения.

Выводы

1. Эволюция программного обеспечения — процесс, дополняющий его разработку. В гибкой методологии разработки эволюция производится разработчиками; в классической модели жизненного цикла эволюция может осуществляться специальной группой (*сопровождение ПО*).
2. Эволюция определяется запросами на изменение. Основные фазы внедрения изменений: оценка влияния; планирование выпуска; реализация изменения.
3. Существует три типа сопровождения: исправление дефектов (в т. ч. упреждающее); адаптация к среде выполнения; внесение новой функциональности.
4. Реинженерия ПО заключается в упрощении структуры программы и / или данных и дополнении документации. Рефакторинг (внесение в программу малых изменений с сохранением функциональности) является упреждающей формой сопровождения ПО во время разработки.

Материалы

Sommerville, Ian

Software Engineering.

Pearson, 2011. — 790 p.

Fowler, Martin

Сайт по рефакторингу.

http://refactoring.com/

Лавріщева К. М.

Програмна інженерія (підручник).

К., 2008. — 319 с.

Спасибо за внимание!