Рефакторинг

Забота о коде

Андрей Скляревский
.NET Developer, Murano Software
Andrew.Sklyarevsky@muranosoft.com

Часто возникающие проблемы с существующим кодом

- Однажды созданная структура не учитывает постоянно возникающих новых требований;
- Заплатки, реализующие требуемую новую функциональность, постепенно заполняют большую часть кода и усложняют поддержку проекта;
- Со временем, для реализации простых функций становится необходимым точное понимание всего кода, что не только тормозит развитие проекта, но и создаёт дополнительные сложности при изменении состава команды разработчиков;

Производительность разработчиков падает

На итерации N количество появляющихся ошибок не оправдывает новые возможности в ПО

Надо переписать весь код?

- МИФ 1: «Было бы время, мы бы переписали всё это как надо»;
- МИФ 2: «Теперь мы знаем точно, что требуется от проекта, поэтому смогли бы создать идеальную архитектуру»;
- МИФ 3: «Привести всё это в порядок за имеющееся время невозможно, поэтому будем писать как есть»;

Надо применять рефакторинг

- Рефакторинг – постепенная работа над кодом, *не влияющая* на функциональность программного обеспечения, однако делающая код более простым, легко читаемым и гибким;
- Рефакторинг предполагает отсутствие изначальной идеальной структуры, однако позволяет приводить её в порядок со временем (и держать её в порядке);
- Необходим вне зависимости от применяемой методологии разработки ПО.

Но зачем?

- Менеджерам рефакторинг может показаться неэффективным процессом с точки зрения производства, т.к. он не приносит ничего нового, не добавляет реализации новой функциональности в ПО, казалось бы, просто отнимает время у разработчиков;
- Если поливать огород и обращать внимание только на то, как растут деревья и овощи, то огород со временем полностью зарастёт сорняками.

Основные признаки кода с «душком»

1. Дублирование кода;

   Copy/Paste

2. Длинные методы;

   100 lines

3. Чрезмерная безопасность;

   Вася не может видеть этот код, т.к. руководство заботится о сохранности кода

4. Большой класс;

5. Много параметров;

```
void DoTheJob(object instance,
int count, int index, JobMode mode,
MainEntity entity, …)
```

6. switch, switch, switch…

```
int typeCode = 0;
switch(typeCode) {
    case 0:
        doJobFor0();
        break;
}
```

Тестирование

- Перед проведением рефакторинга *очень хорошо* написать unit test, который позволит убедиться в том, что после рефакторинга код работает *точно так же*, как и перед его проведением;
- Код, который можно протестировать только используя GUI (или Веб-интерфейс) скорее всего вызовет больше проблем, чем полностью самотестирующийся код;
- Тестирования много не бывает.

// Комментарии

- Код, содержащий много комментариев часто изобилует дурным «душком», а комментарии выступают в роли дезодоранта, облегчая понимание кода читателю;
- Обычно, можно использовать рефакторинг, чтобы сделать код понятным практически без комментариев.

Приёмы рефакторинга

- Сокращение размеров методов при помощи выделения отдельных блоков в более маленькие методы, с простым и понятным назначением;
- Обязательная инкапсуляция полей;
- Сокращение размеров классов путём выделения отдельных групп методов и полей в отдельные классы или подклассы;
- Сокращение размеров иерархии при помощи перемещения методов и полей, выделения родительских классов и т.д.

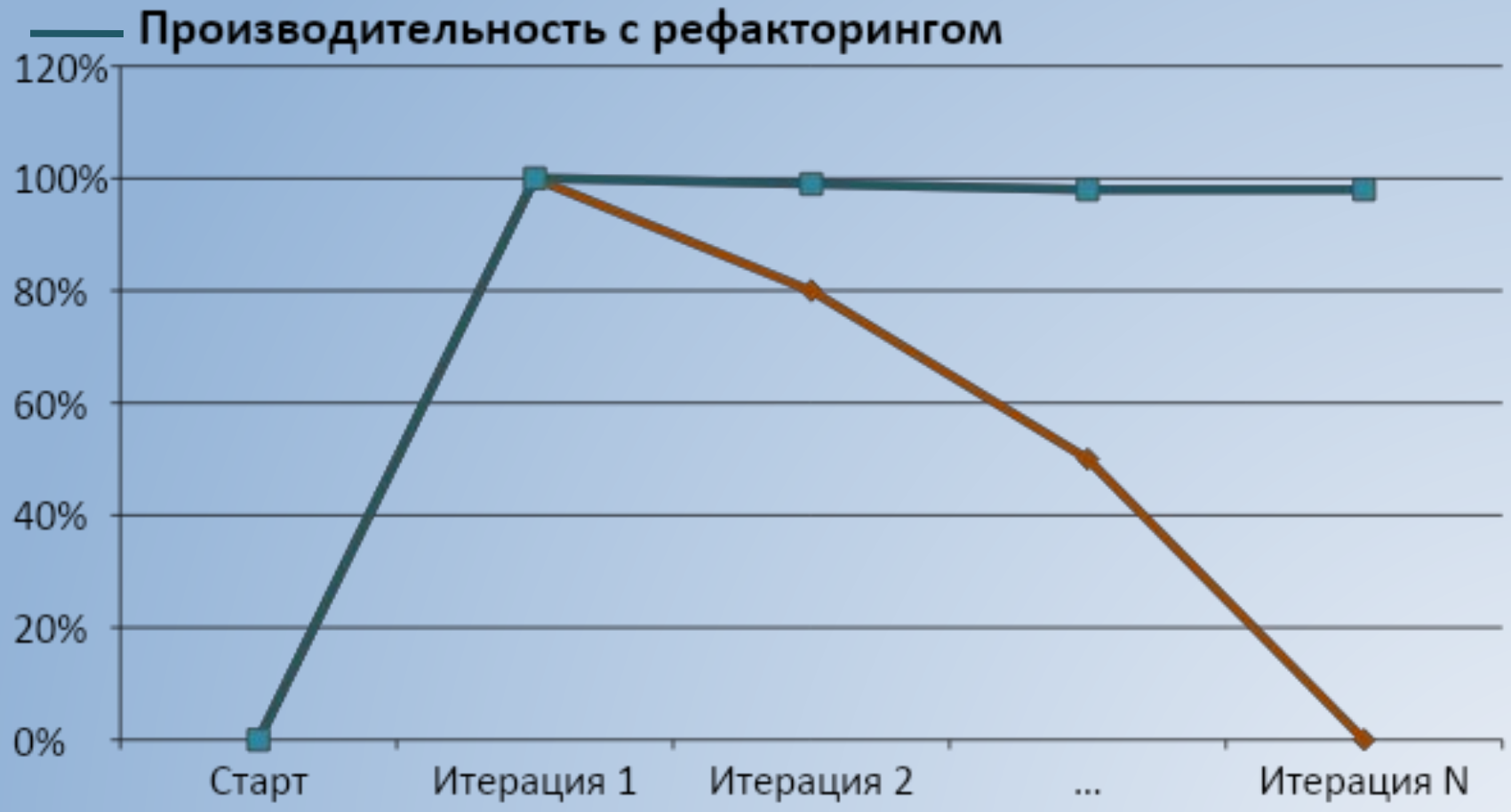
Глобальный рефакторинг

- Иногда разработчики начинают проводить рефакторинг, углубляются в дебри кода, оказывается что он ещё более несовершенен и пытаются что-то с этим сделать, в результате создавая ещё более сильный беспорядок, который отказывается компилироваться;
- Нельзя вводить понятие «глобального» рефакторинга, т.к. рефакторинг есть постепенное улучшение кода, которым нужно заниматься постоянно.

Есть время на рефакторинг?

- МИФ: «Я занимаюсь рефакторингом, когда есть время, однако он не эффективно отражается на процессе, поэтому я стараюсь не уделять ему слишком много времени»;
- *Рекомендуется заниматься рефакторингом на постоянной основе.*

К чему стремимся?



Ссылки

(по просьбе участников user group)

- Лучшая книга про заботу о коде – **«Рефакторинг»** Мартина Фаулера; http://www.martinfowler.com/books.html#refactoring
- Хороший add-in для Visual Studio, с поддержкой рефакторинга: **JetBrains ReSharper**; http://www.jetbrains.com/resharper/