sourcesense

The Birthday Greetings Kata

Matteo Vaccari
m.vaccari@sourcesense.com

The problem

1. Read employee records from a file
2. Filter employees whose birthday is today
3. Send a personalized greetings message by email

```
public void sendGreetings(String fileName, OurDate ourDate,
        String smtpHost, int smtpPort) throws IOException, ... {

    BufferedReader in = new BufferedReader(new FileReader(fileName));
    String str = "";
    str = in.readLine(); // skip header
    while ((str = in.readLine()) != null) {
        String[] employeeData = str.split(", ");
        Employee employee = new Employee(employeeData[1], employeeData[0],
                employeeData[2], employeeData[3]);
        if (employee.isBirthday(ourDate)) {
            String body = "Happy Birthday, dear %NAME%!".replace("%NAME%",
                    employee.getFirstName());
            String subject = "Happy Birthday!";
            Greetings greetings = new Greetings(employee, subject, body);
            sendMessage(smtpHost, smtpPort, "sender@here.com", greetings);
        }
    }
}
```

```
protected void sendMessage(String smtpHost, int smtpPort, String sender,
        Greetings greetings) throws AddressException, MessagingException {

    // Create a mail session
    java.util.Properties props = new java.util.Properties();
    props.put("mail.smtp.host", smtpHost);
    props.put("mail.smtp.port", "" + smtpPort);
    Session session = Session.getDefaultInstance(props, null);

    // Construct the message
    Message message = new MimeMessage(session);
    message.setFrom(new InternetAddress(sender));
    message.setRecipient(Message.RecipientType.TO, new InternetAddress(
            greetings.recipientEmail()));
    message.setSubject(greetings.subject());
    message.setText(greetings.body());

    // Send the message
    Transport.send(message);
}
```

```
public static void main(String[] args) {
    BirthdayService service = new BirthdayService();
    try {
        service.sendGreetings("employee_data.txt",
                new OurDate("2008/10/08"), "localhost", 25);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Our goals

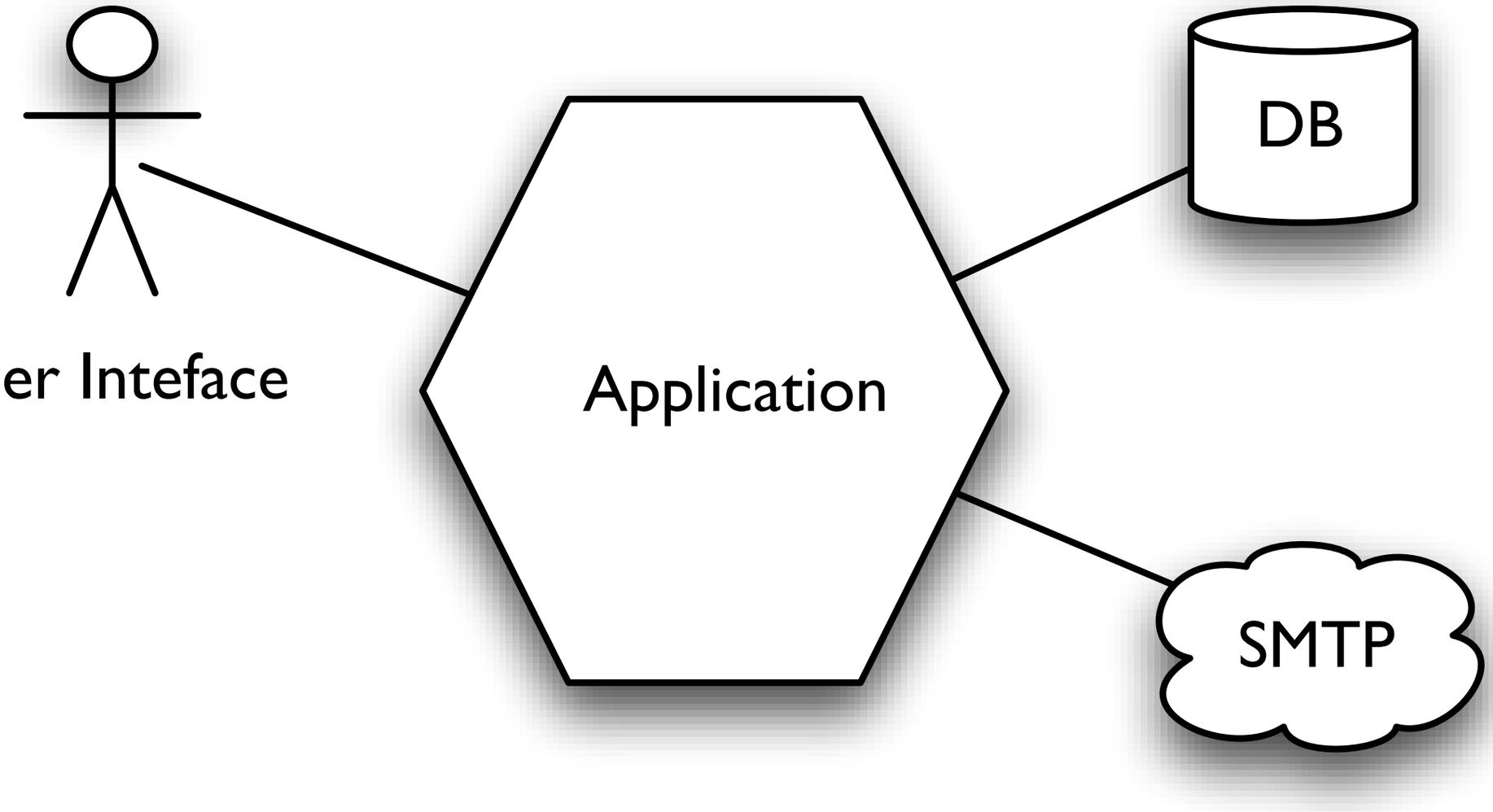
- Code that *does not cost a lot* to change when the requirements change
- Code that can is easy to *test*
- Code that is easy to *understand*

Business logic must not depend on low-level APIs

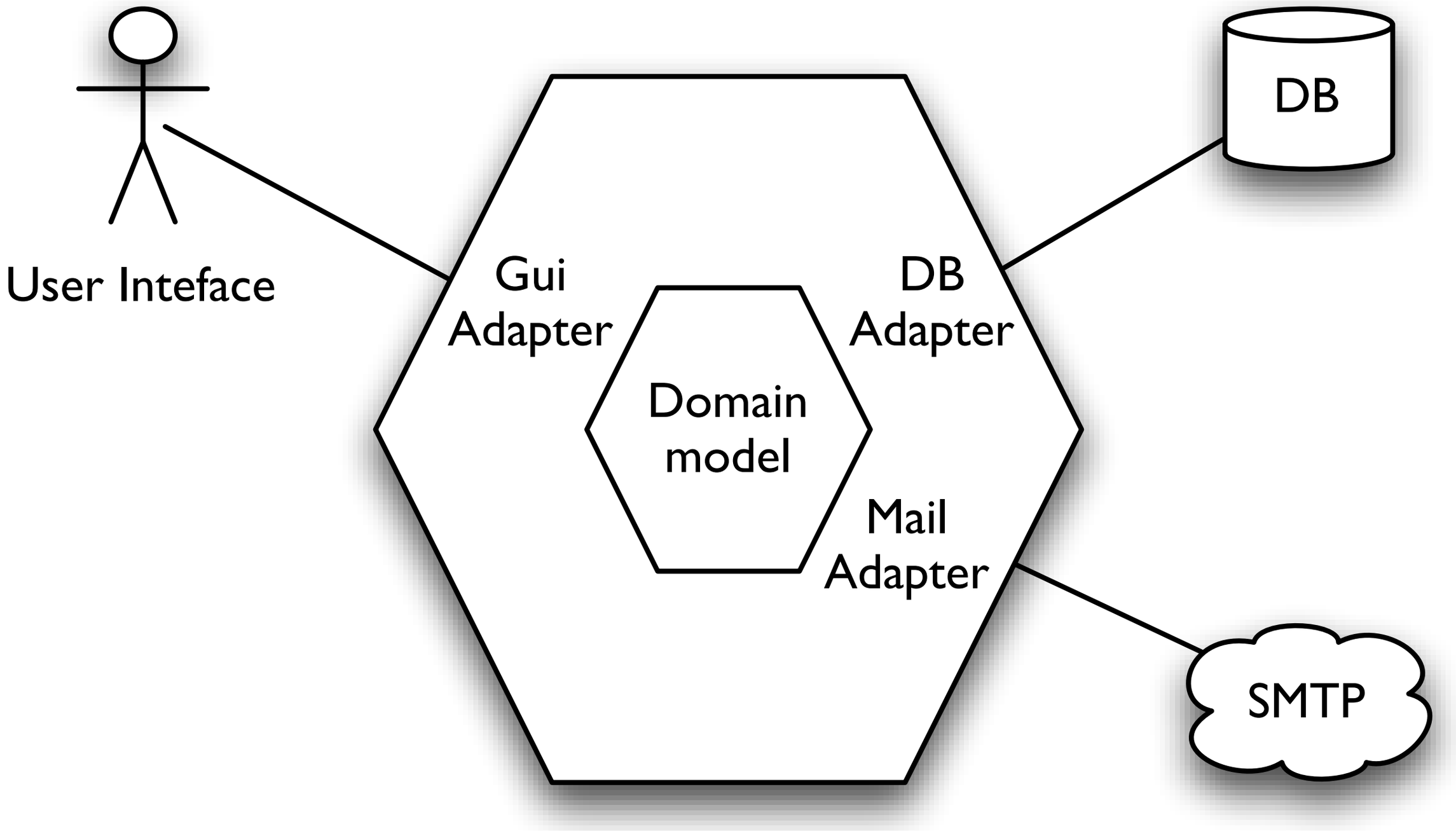
Code that talks with external systems is clearly separated

Unit tests must be very fast and reliable

The hexagonal architecture

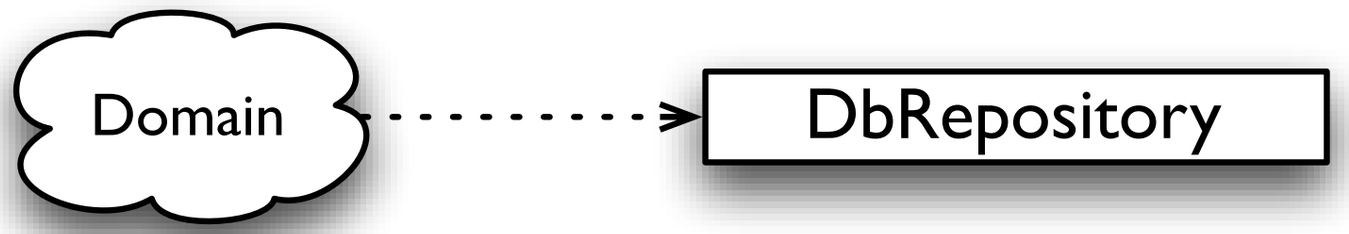


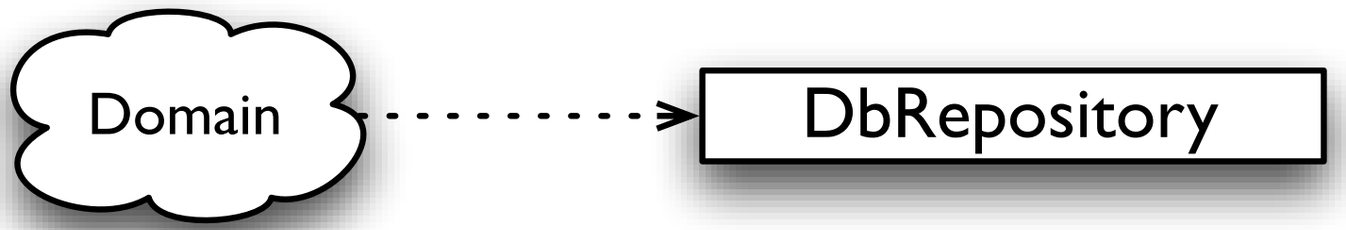
Term coined by Alistair Cockburn

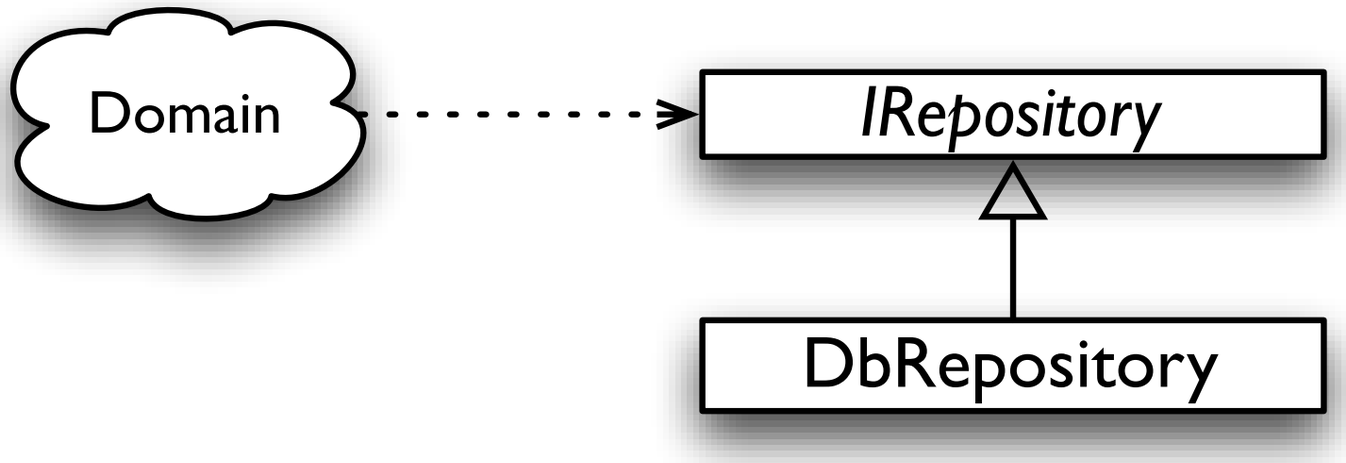

User Inteface
Gui
Adapter
Domain
model
DB
Adapter
Mail
Adapter
DB
SMTP

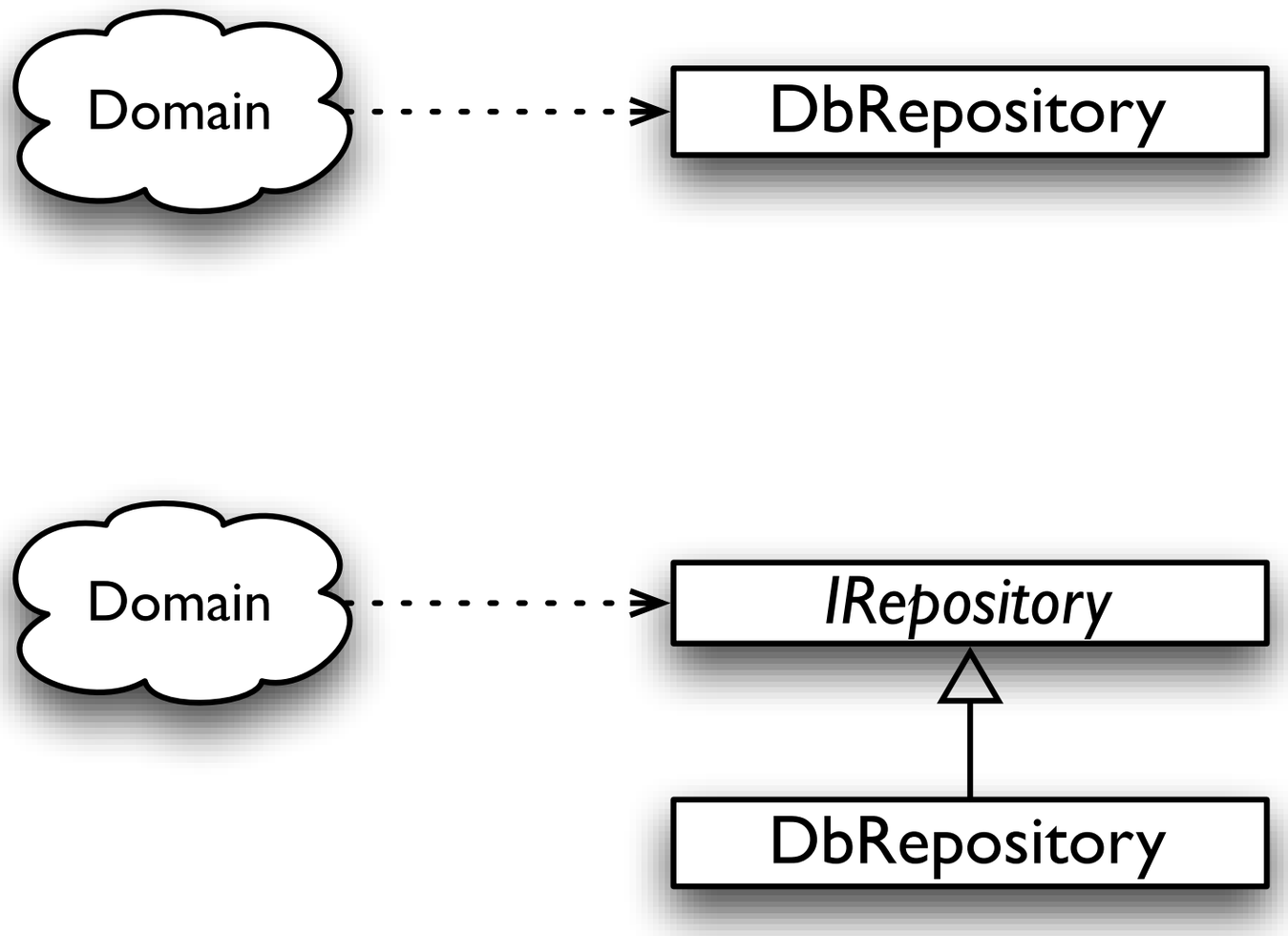
In the hexagonal architecture

- Domain Model depends on nothing
- Everything depends on the domain model

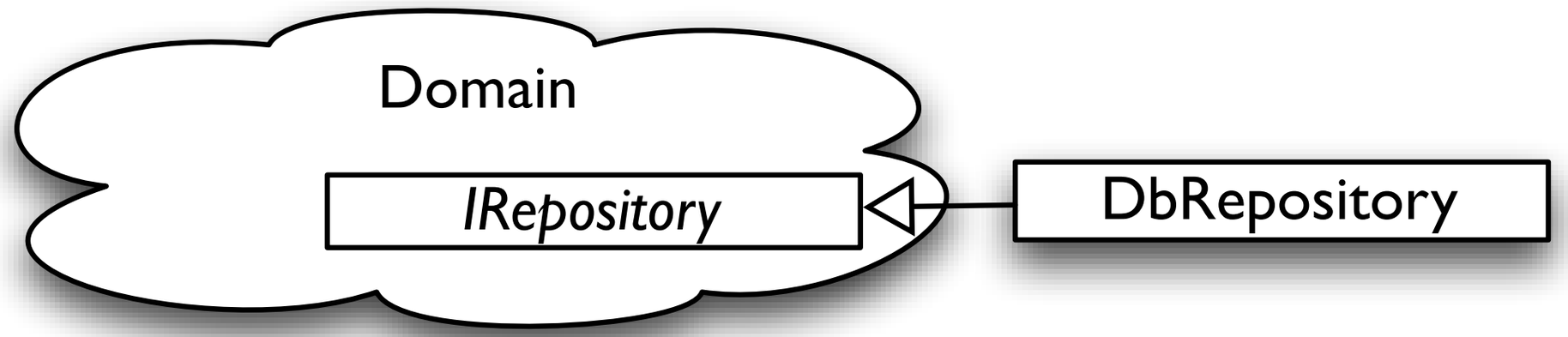

Domain
DbRepository


Domain
DbRepository


Domain
IRepository
DbRepository


Domain
DbRepository

Domain
IRepository
DbRepository


Domain
IRepository
DbRepository

Refactoring technique

Refactoring technique

- Small steps

Refactoring technique

- Small steps
- Keep the tests green at all times

Refactoring technique

- Small steps
- Keep the tests green at all times
- ***Small*** steps!

Refactoring technique

- Small steps
- Keep the tests green at all times
- ***Small*** steps!
- Keep the tests ***green at all times!***

A refactoring move: Extract class

1. Identify a responsibility
2. Extract a method
3. Create a new class
4. Move that method to the new class

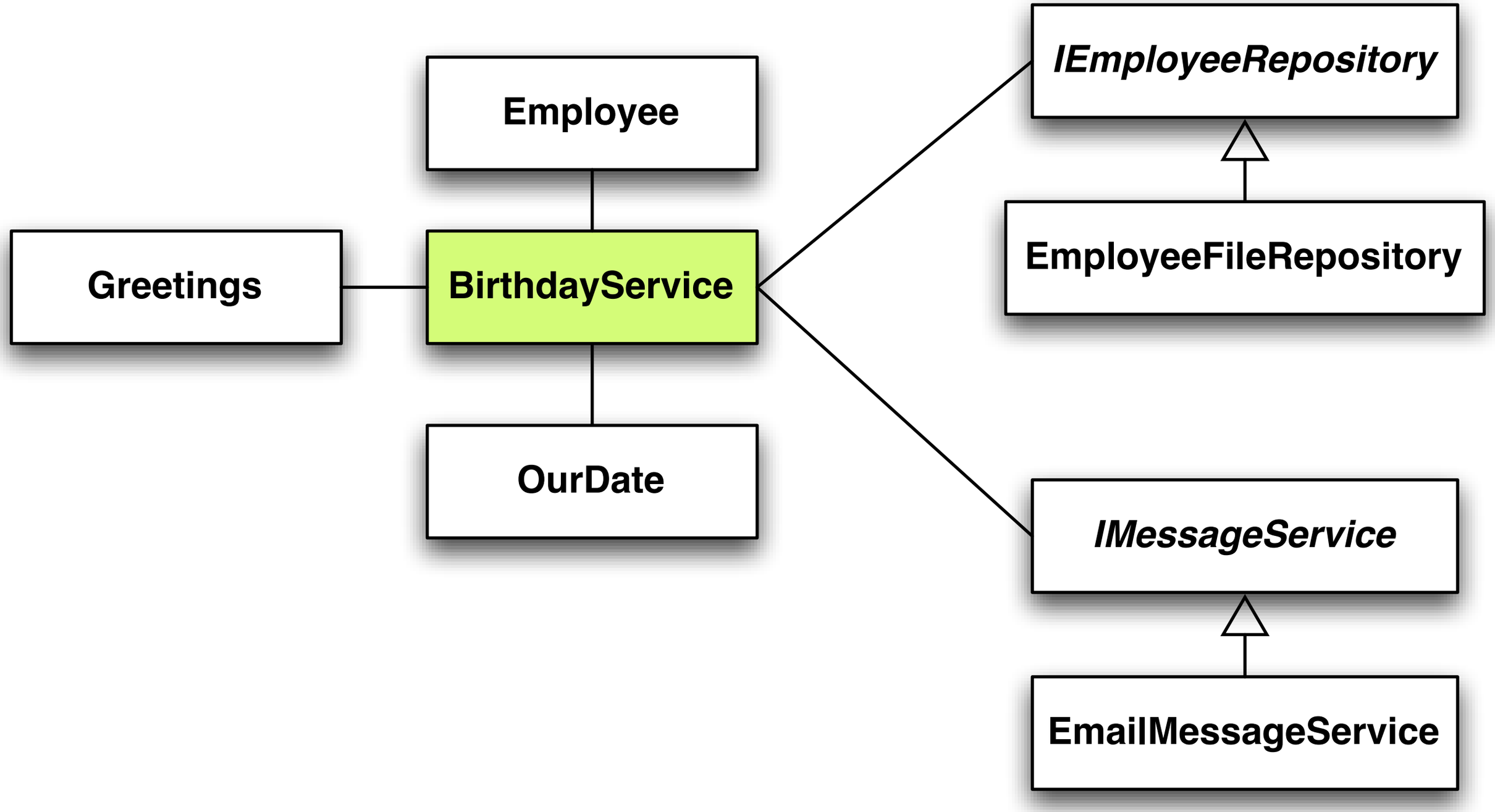
Where we'd like to be

```
public class BirthdayService {

  private EmployeeRepository repository;
  private MessageService messageService;

  public BirthdayService(IEmployeeRepository repository, IMessageService messageService) {
    this.repository = repository;
    this.messageService = messageService;
  }

  public void sendGreetings(OurDate today) {
    List<Employee> employees = repository.findEmployeesWhoseBirthdayIs(today);
    for (Employee employee: employees) {
      Greetings greetings = new Greetings(employee);
      messageService.send(greetings);
    }
  }
}
```


Employee
IEmployeeRepository
Greetings
BirthdayService
EmployeeFileRepository
OurDate
IMessageService
EmailMessageService

A possible roadmap

1. Make sure the tests run
2. Get familiar with the code
3. Extract the Greetings class
4. Extract the MessageService class
5. Extract the EmployeeRepository class

References

code:
git://github.com/xpmatteo/birthday-greetings-kata.git

documentation:
http://matteo.vaccari.name/blog/archives/154