بسمه تعالی

# درس مهندسی نرم‌افزار پیشرفته

## فصل شانزدهم

# توسعه مبتنی بر جنبه ( *Aspect-oriented Development*)

دکتر فریدون شمس

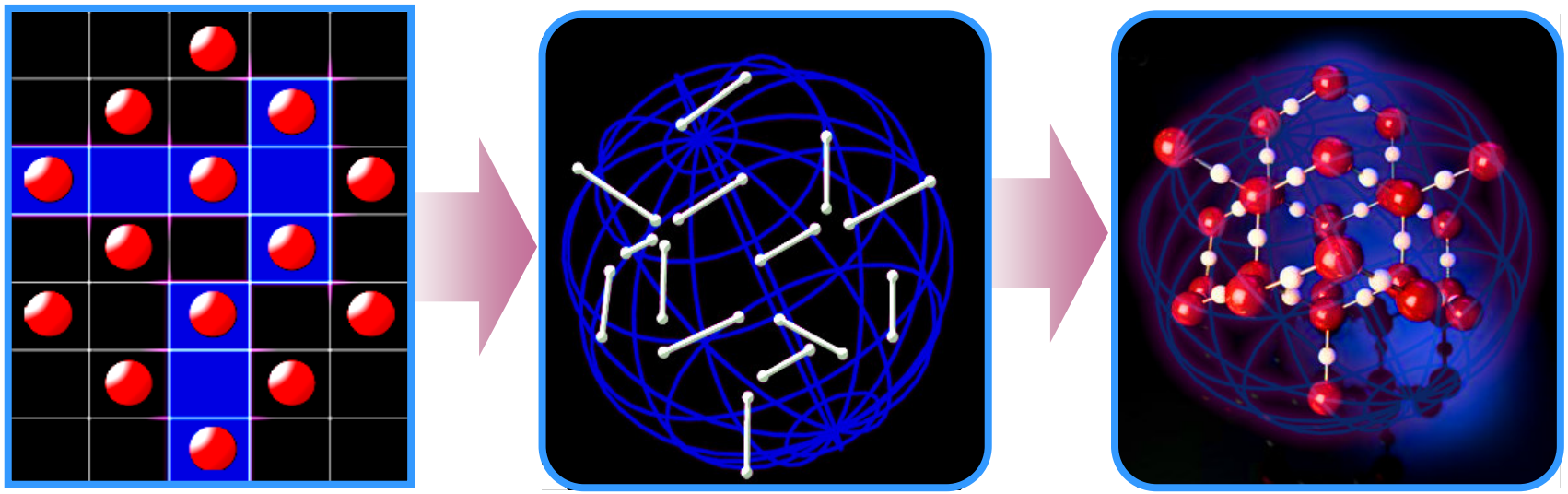
# اهداف جلسه

- ذاتی بودن پیچیدگی نرم‌افزار
- توسعه نرم‌افزار مبتنی بر جنبه
- مفاهیم توسعه مبتنی بر جنبه
- انواع جنبه‌ها و دسته‌بندی آنها
- توسعه نرم‌افزار مبتنی بر جنبه

# فهرست مطالب

- پیچیدگی ذاتی نرم‌افزار
- توسعه نرم‌افزارهای مدرن و مشکلات آن
- مفاهیم توسعه مبتنی بر جنبه
- نمونه‌های توجه به جنبه
- تفاوت شی با جنبه
- توسعه نرم‌افزار مبتنی بر جنبه

# توسعه نرم‌افزار

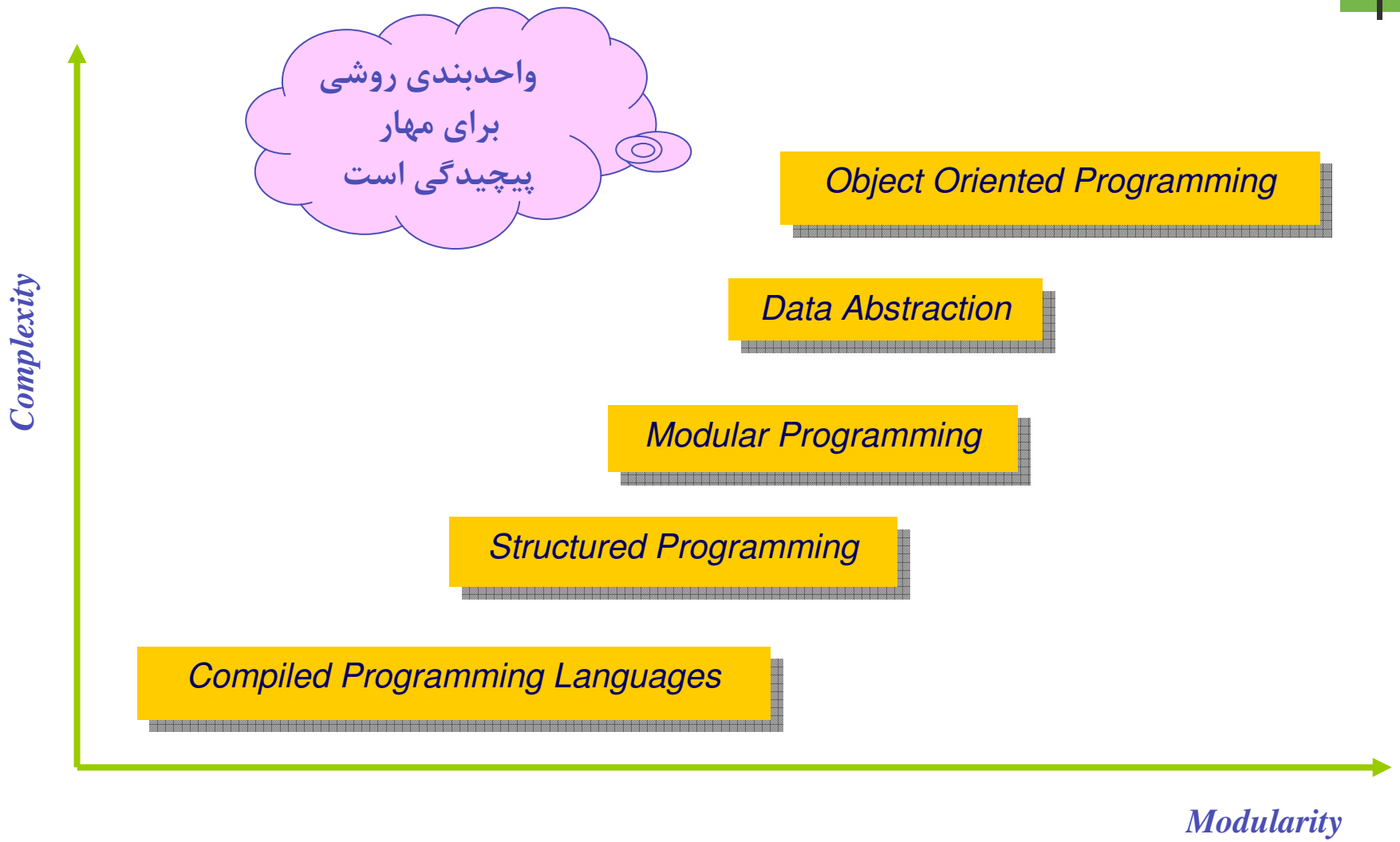


نرم‌افزارهای قدیمی | نرم‌افزارهای موجود | نرم‌افزارهای آینده

**توسعه نرم‌افزار از حالت ثابت، محاسباتی و داده‌گرا**

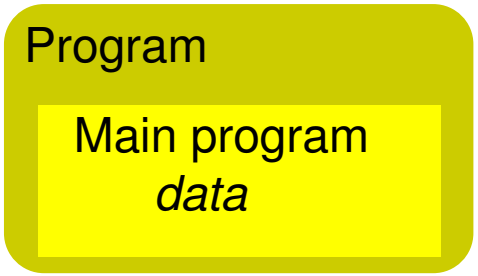
**به حالت پویا، بسته به نیاز و مولفه‌گرا تبدیل شده است**

# توسعه نرم‌افزار (ادامه)



# توسعه نرم‌افزار (ادامه)

## برنامه‌نویسی ساختاریافته

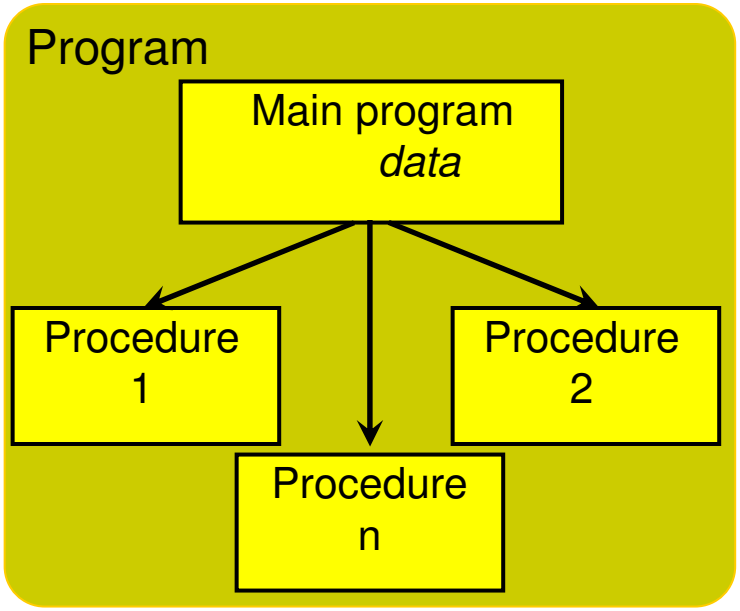



```
i = 1
while (i < 4) {
  print(i)
  i = i + 1
}
```

- راحتی خواندن و نوشتن
- قابلیت توسعه پائین
- قابلیت نگهداری پائین
- قابلیت استفاده مجدد پائین
- فاقد واحدبندی

# توسعه نرم‌افزار (ادامه)

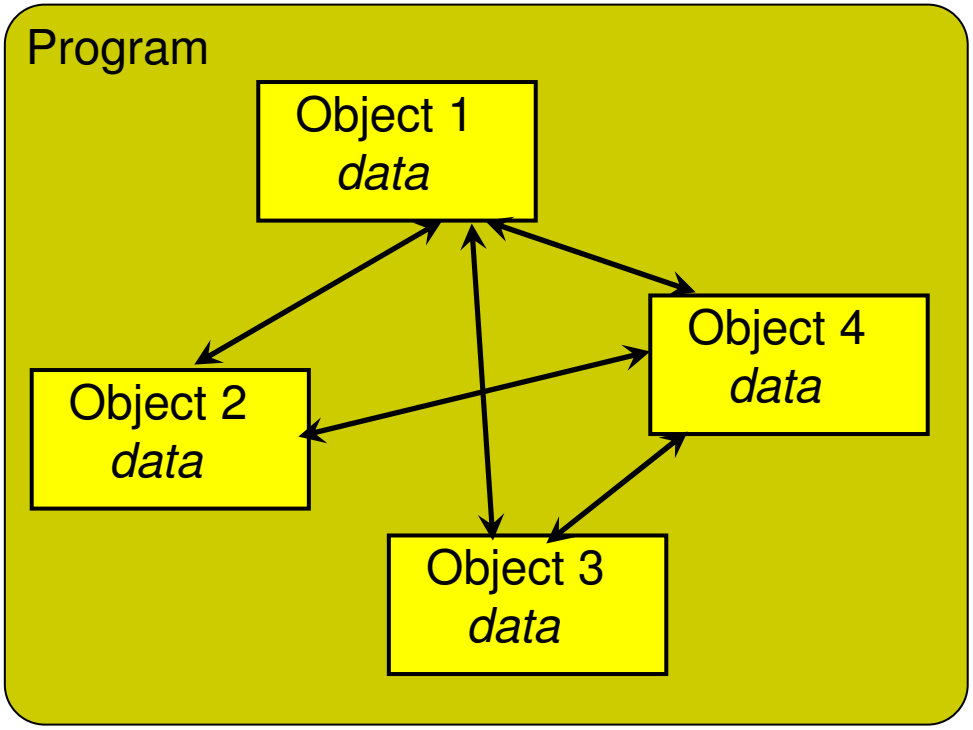
## برنامه‌نویسی رویه‌ای




- راحتی خواندن و نوشتن
- قابلیت توسعه بهبودیافته
- قابلیت نگهداری بهبودیافته
- قابلیت استفاده‌مجدد بهبودیافته
- واحدبندی بهبودیافته

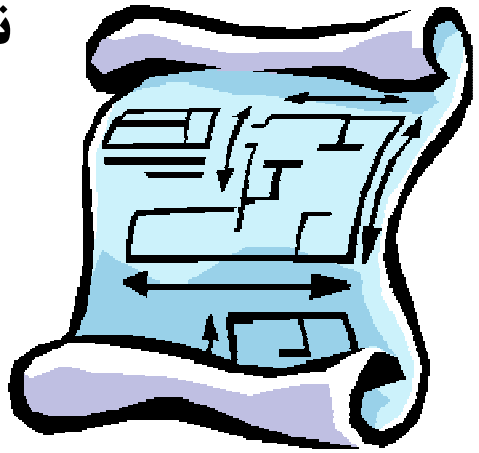
# توسعه نرم‌افزار (ادامه)

## برنامه‌نویسی شی‌گرا




- راحتی خواندن و نوشتن
- قابلیت توسعه خوب
- قابلیت نگهداری خوب
- قابلیت استفاده‌مجدد خوب
- واحدبندی خوب

# توسعه نرم‌افزار (ادامه)



نیازهای وظیفه‌مندی

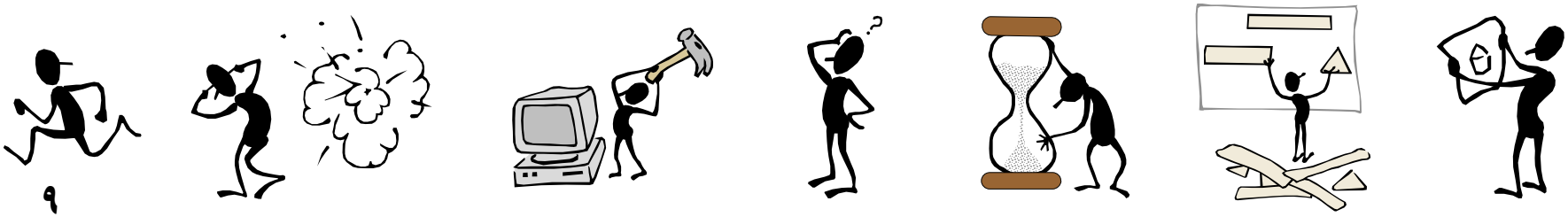
+

نیازهای غیروظیفه‌مندی

+

نیازمندی‌های توسعه نرم‌افزار

پیچیدگی



# توسعه نرم‌افزار (ادامه)

- مشکلات روش‌های شی‌گرا از آنجا ناشی می‌شود که

**توسعه مبتنی بر جنبه (*Aspect-oriented*) با نگاهی بر نیازمندی‌ها و پوشش بهینه آنها ارائه شده است**

- تمرکز شی‌گرایی بر روی بهبود برنامه‌نویسی و کدها است، در حالیکه نیاز است تا تمرکز بر روی نیازمندی‌ها باشد
  - چالش اصلی توسعه نرم‌افزار، برآورده‌سازی نیازمندی‌هاست

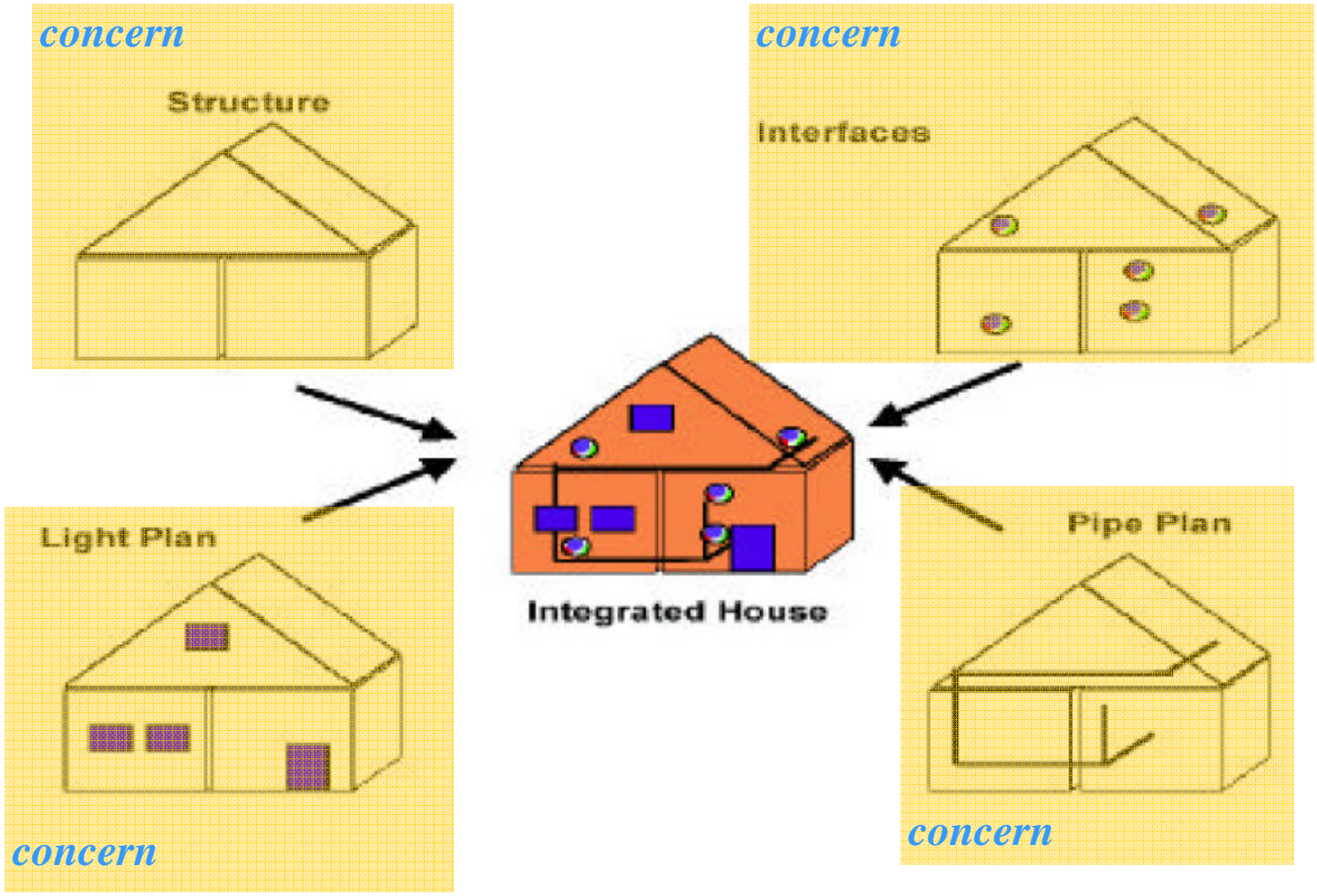
# مفاهیم توسعه مبتنی بر جنبه

- دغدغه (*Concern*)
  - نیازمندی سیستم که برای ذینفعان اولویت دارد
  - مجموعه خاصی از رفتارهای مورد نیاز برای یک برنامه
  - بر ماژول‌های مختلف نرم‌افزار تاثیر می‌گذارد
  - می‌توانند وظیفه‌مندی یا غیروظیفه‌مندی باشند
    - *Business logic ،Performance ،Logging and Debugging*
  - می‌توانند سطح بالا یا سطح پائین باشند
    - *Caching ،Security, QoS*

# مفاهیم توسعه مبتنی بر جنبه (ادامه)

- جداسازی دغدغه‌ها (*Separation of concerns*) روشی برای توجه به آنها به صورت مجزاست که اجازه رهگیری شفاف نیازمندی‌ها به پیاده‌سازی را می‌دهد
- جداسازی در روش‌های سنتی از طریق واحدبندی و کپسوله‌سازی فراهم می‌گردد
  - زبان‌های رویه‌ای دغدغه‌ها را به رویه‌ها تبدیل می‌کنند
  - زبان‌های شی‌گرا دغدغه‌ها را اشیاء تبدیل می‌کنند
  - طراحی‌های سرویس‌گرا دغدغه‌ها را به سرویس تبدیل می‌کنند

# مفاهیم توسعه مبتنی بر جنبه (ادامه)

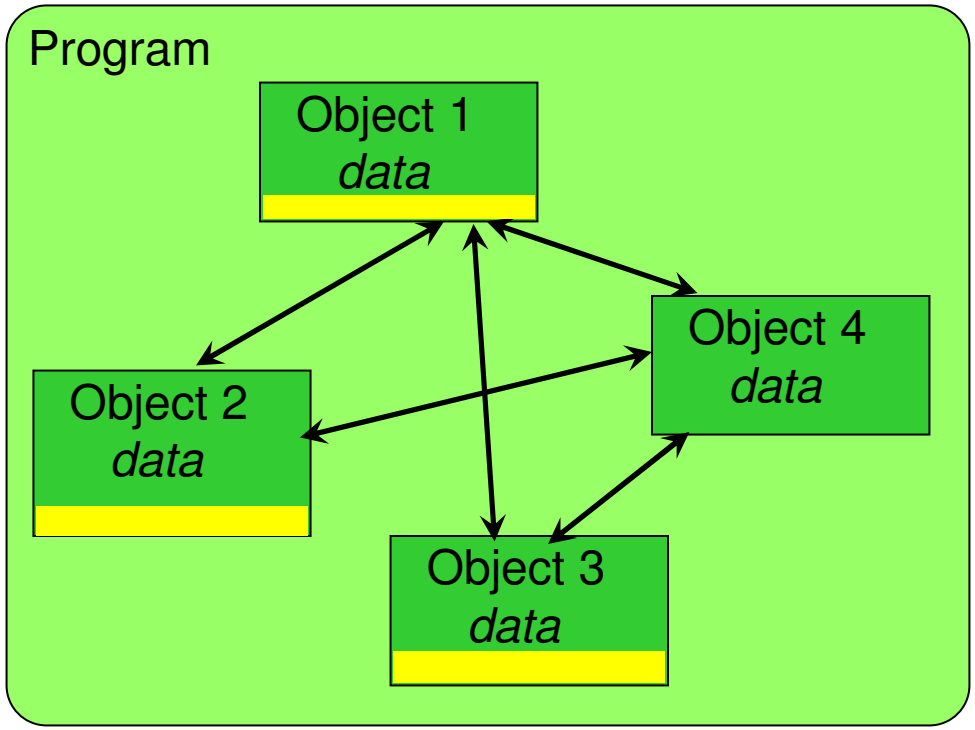



*Separation of concerns*

# مفاهیم توسعه مبتنی بر جنبه (ادامه)

- *Core concerns*
  - دغدغه‌هایی هستند که به اهداف اصلی سیستم مرتبط هستند و معمولاً در یک رویه، ماژول یا شی قرار می‌گیرند
- *Crosscutting Concerns*
  - بطور معمول در چندین رویه، ماژول یا شی قرار می‌گیرند
  - در هنگام تغییر به دو صورت مشکل ایجاد می‌شود
    - *Tangling* : یک مولفه بیش از یک نیازمندی را پیاده‌سازی می‌کند
    - *Scattering* : پیاده‌سازی یک نیازمندی توسط چند مولفه انجام می‌شود

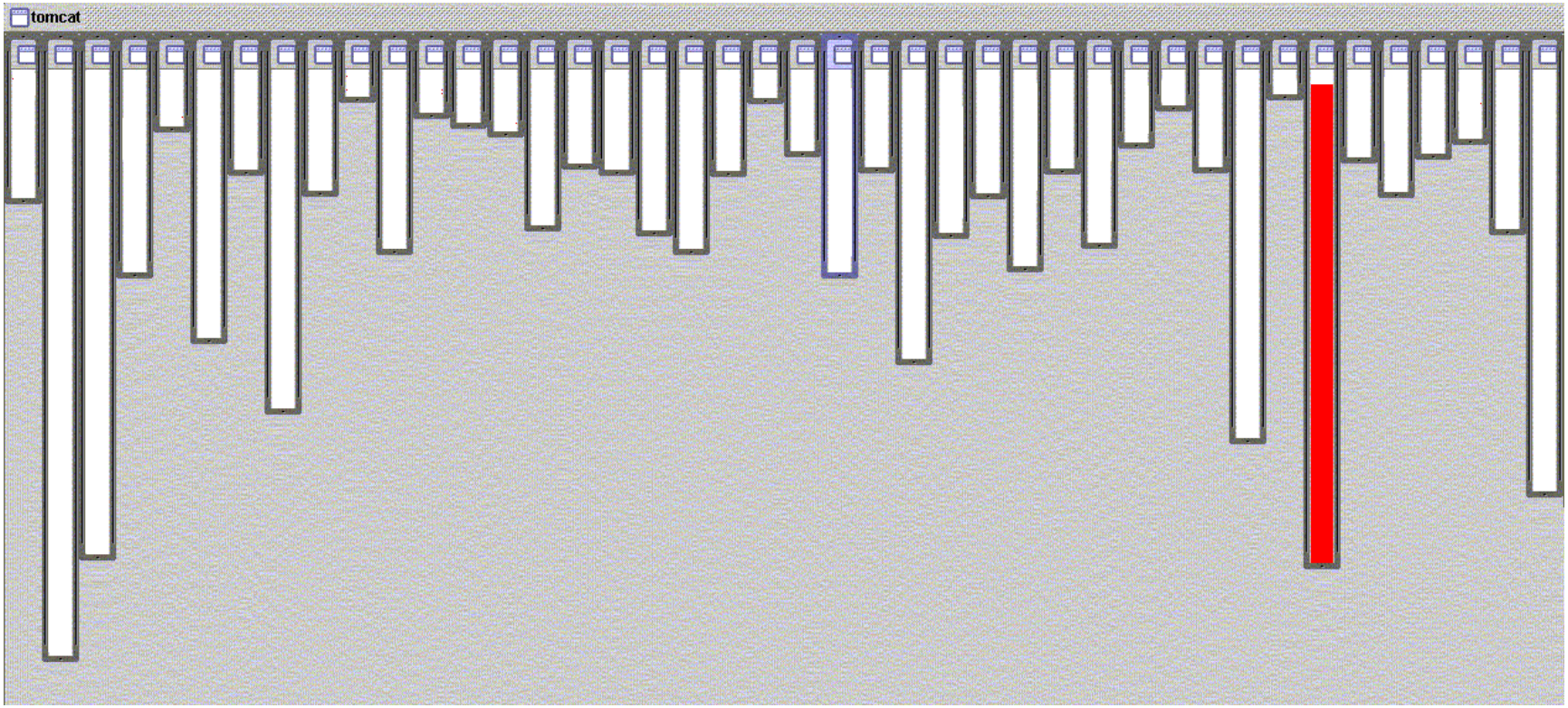
# مفاهیم توسعه مبتنی بر جنبه (ادامه)




| Concern | Implementation |
|---|---|
| A | Object 1 |
| B | Object 2 |
| C | Object 3 |
| D | Object 4 |
| E | Object 1,2,3 |

***Crosscutting Concerns***

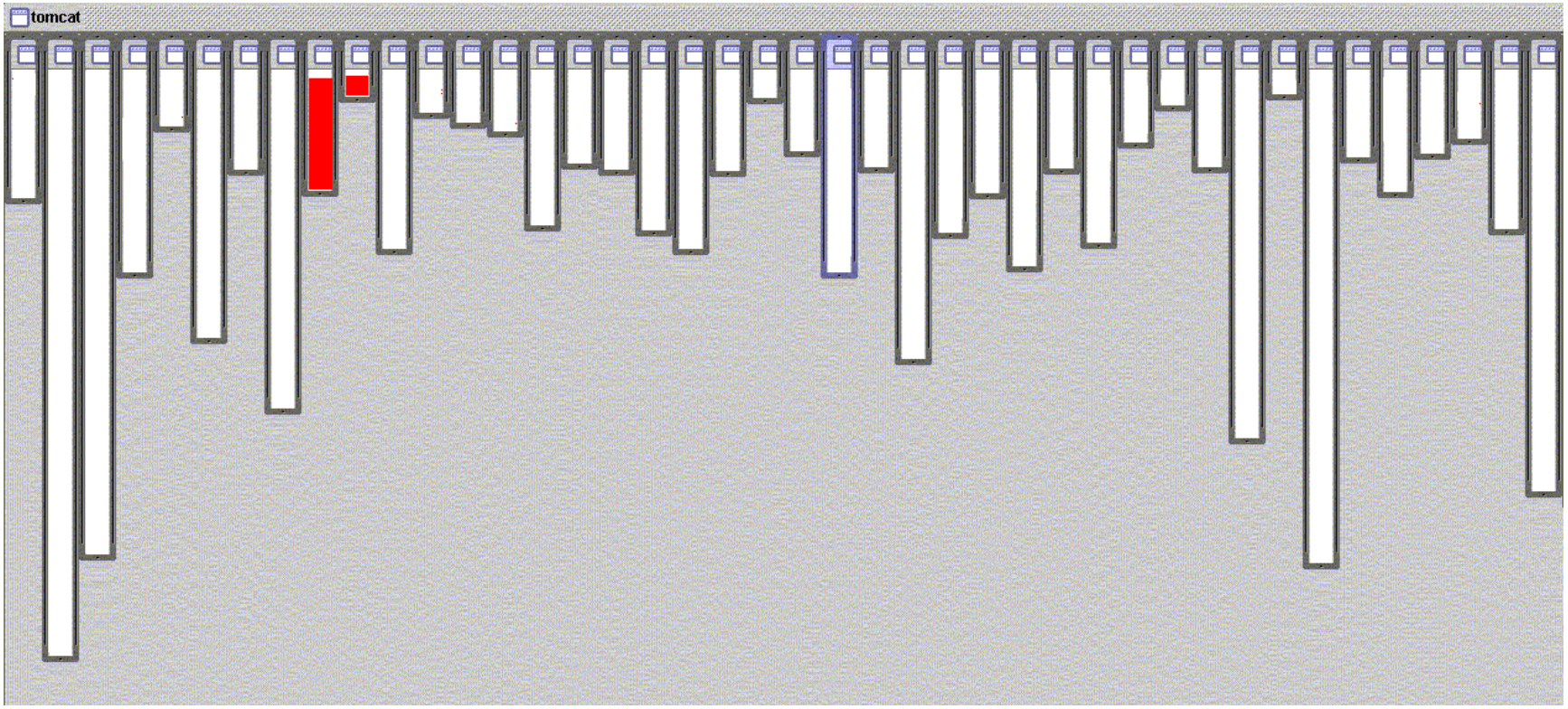
# نمونه توجه به جنبه




*Good modularization*

*XML parsing is implemented in its own module*

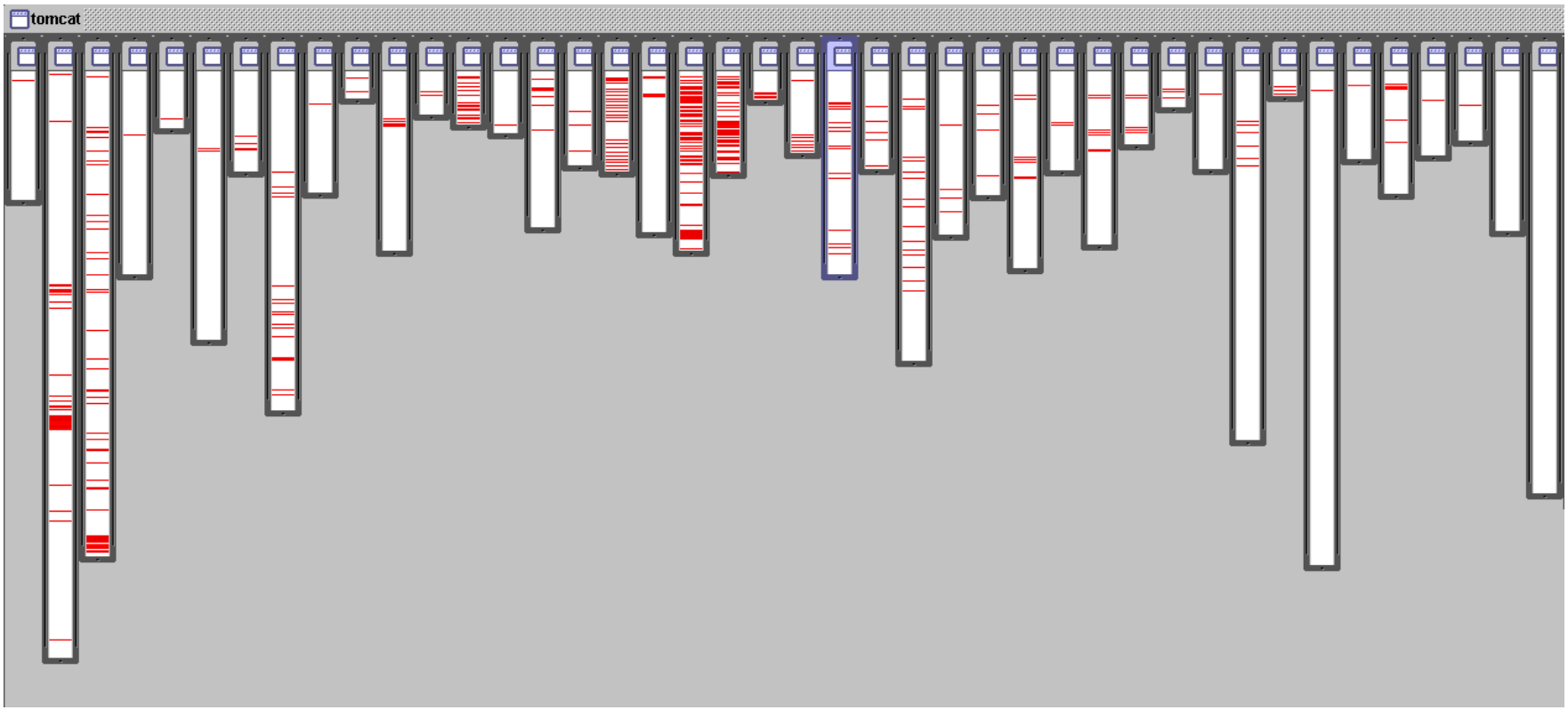
# نمونه توجه به جنبه (ادامه)




*Good modularization*

*URL pattern matching is implemented in 2 modules*

# نمونه توجه به جنبه (ادامه)




***Bad modularization***

***Logging is implemented in a lot of different places***

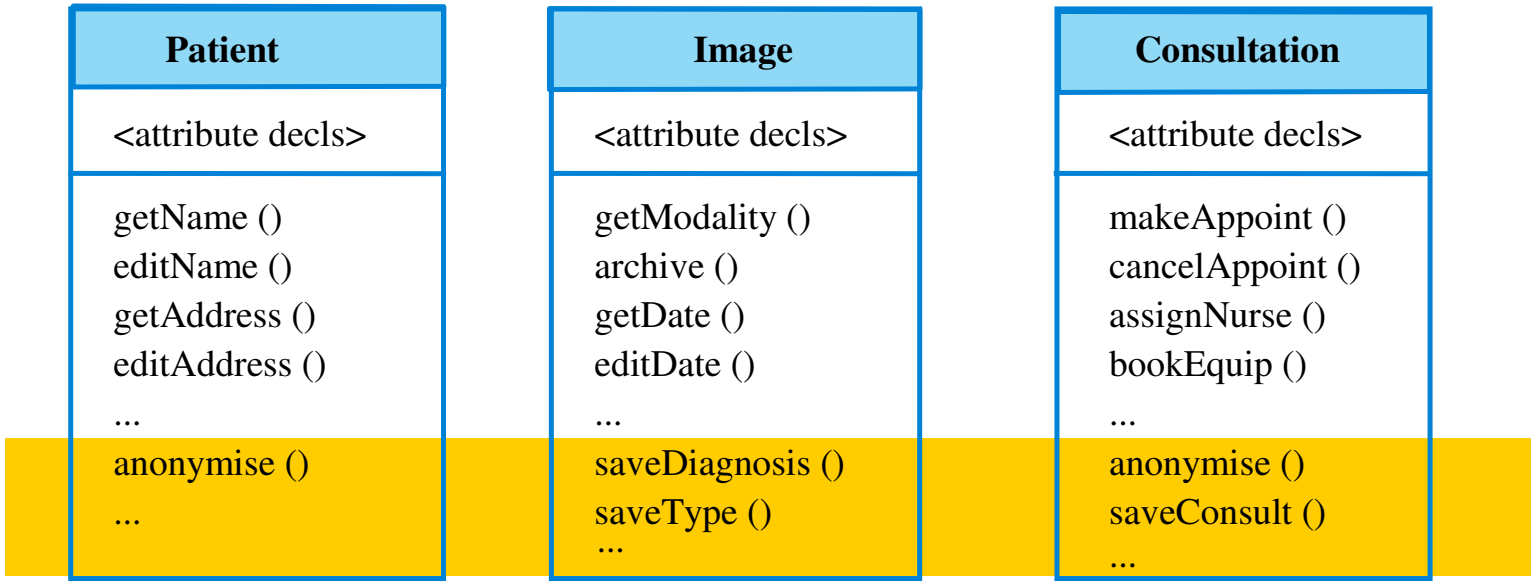
# مفاهیم توسعه مبتنی بر جنبه (ادامه)

## درهم پیچیدگی (*Tangling*)

```
synchronized void put (SensorRecoed rec) throws Interrupted Exception
 {
  if (numberOfEntries == bufsize)
       wait ();
  store [back] = new SensorRecord (rec.sensorId, rec.sensoral) ;
   back = back + 1;
   if (back == bufsize)
       back =0;
  numberOfEntries = numberOfEntries +1;
  notify();
 } // put
```

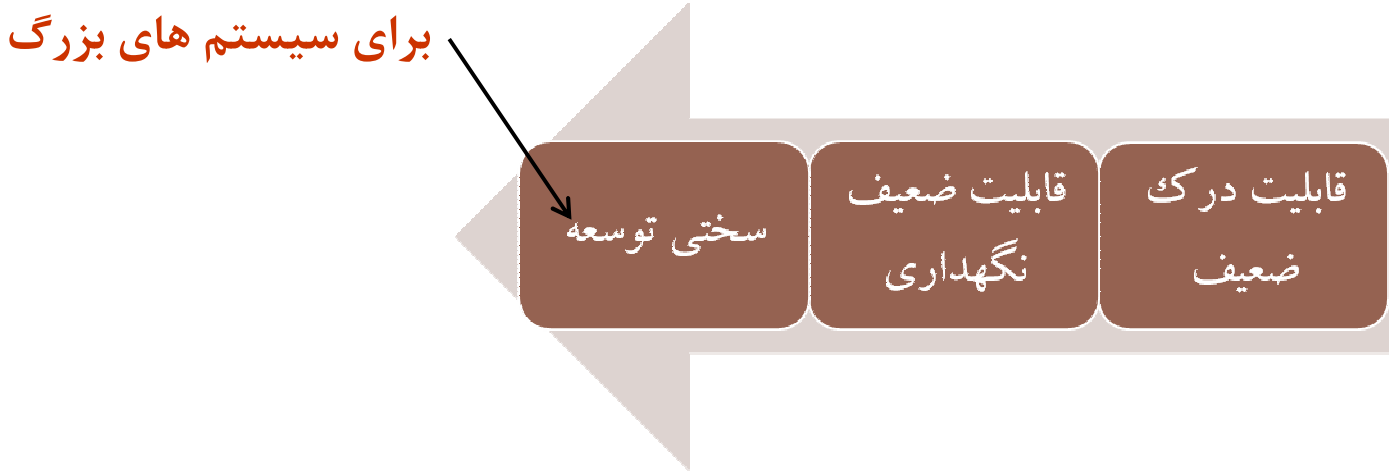
# مفاهیم توسعه مبتنی بر جنبه (ادامه)

## پراکندگی (*Scattering*)



# مفاهیم توسعه مبتنی بر جنبه (ادامه)

- پیامدهای درهم پیچیدگی و پراکندگی
  - تغییر در یک دغدغه موجب تغییر در دغدغه‌های دیگر می شود
  - عدم کشف نیازمندی‌های هر دغدغه از طریق کد منبع
  - از بین بردن واحدبندی سیستم
  - ضعف در درک و فهم



# مفاهیم توسعه مبتنی بر جنبه – انواع دغدغه‌ها

- کارکردی (*Functional*)
  - با وظیفه‌مندی خاصی از سیستم مرتبط هستند
- کیفیت سرویس (*Quality of Service*)
  - رفتار غیروظیفه‌مندی سیستم مرتبط هستند
- سیاست (*Policy*)
  - سیاست‌هایی که بر استفاده از سیستم حاکم هستند
- سیستم (*System*)
  - به خصوصیات سیستم به‌عنوان یک کل مانند قابلیت نگهداری و پیکربندیش مرتبط هستند
- سازمانی (*Organisational*)
  - به اهداف سازمانی و اولویت‌های آن مرتبط هستند

# مفاهیم توسعه مبتنی بر جنبه (ادامه)

- جنبه (*Aspect*)
  - تجریدی است که کد مرتبط با *cross-cutting concerns* را کپسوله می‌کند
  - در هر جنبه اطلاعاتی در مورد اینکه چه وقت باید به کد برنامه اضافه شود، وجود دارد
  - یک جنبه متشکل از دو بخش *Advice Code*, *Pointcut* است
  - جایی در برنامه که جنبه می‌تواند در آنجا افزوده شود، نقطه اتصال (*Joint Point*) نام دارد
  - به مجموعه‌ای از نقاط اتصال در برنامه *Pointcut* گویند

# مفاهیم توسعه مبتنی بر جنبه (ادامه)

- *Advice*
  - پیاده‌سازی کد یک دغدغه
- *Waving*
  - تلفیق کد *Advice* در برنامه با استفاده از *Aspect Weaver*
- *Joint Point Model*
  - مجموعه‌ای از رویدادها که می‌توانند در یک *Pointcut* مورد اشاره قرار گیرند

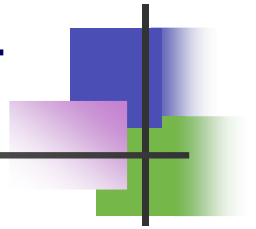
# مفاهیم توسعه مبتنی بر جنبه (ادامه)

- انواع محل‌های قرارگیری JP
  - **متدها**: فراخوانی و اجرای متد دو تا JP محسوب می‌شوند
  - **سازنده‌ها**: فراخوانی و اجرای یک سازنده، نوعی از JP ها است
  - **استثناءها**: رخ دادن استثناء و اجرا رفتار مورد انتظار- دو نقطه اصلی در اجرای یک برنامه کاربردی- نوعی از JP ها است
  - **فیلدها**: عملیات خواندن و نوشتن بر روی فیلدها می‌توانند به عنوان JP در نظر گرفته شوند

# نمونه Aspect

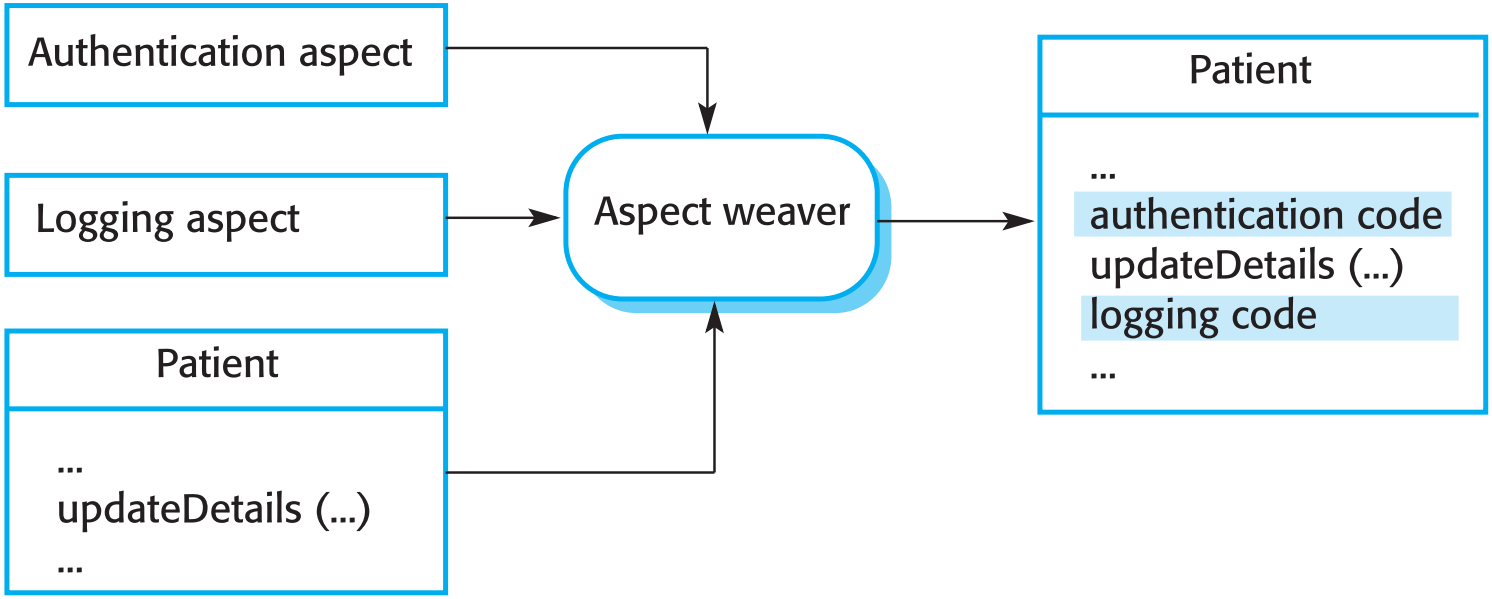
```
aspect authentication
{
      before: call (public void update* (..))   // this is a pointcut
      {
// this is the advice that should be executed when woven into
// the executing system
            int tries = 0 ;
string userPassword = Password.Get ( tries ) ;
            while (tries < 3 && userPassword != thisUser.password ( ) )
            {
                // allow 3 tries to get the password right
                tries = tries + 1 ;
                userPassword = Password.Get ( tries ) ;
            }
            if (userPassword != thisUser.password ( )) then
                //if password wrong, assume user has forgotten to logout
                System.Logout (thisUser.uid) ;
      }
} // authentication
```

# *Aspect weaving*



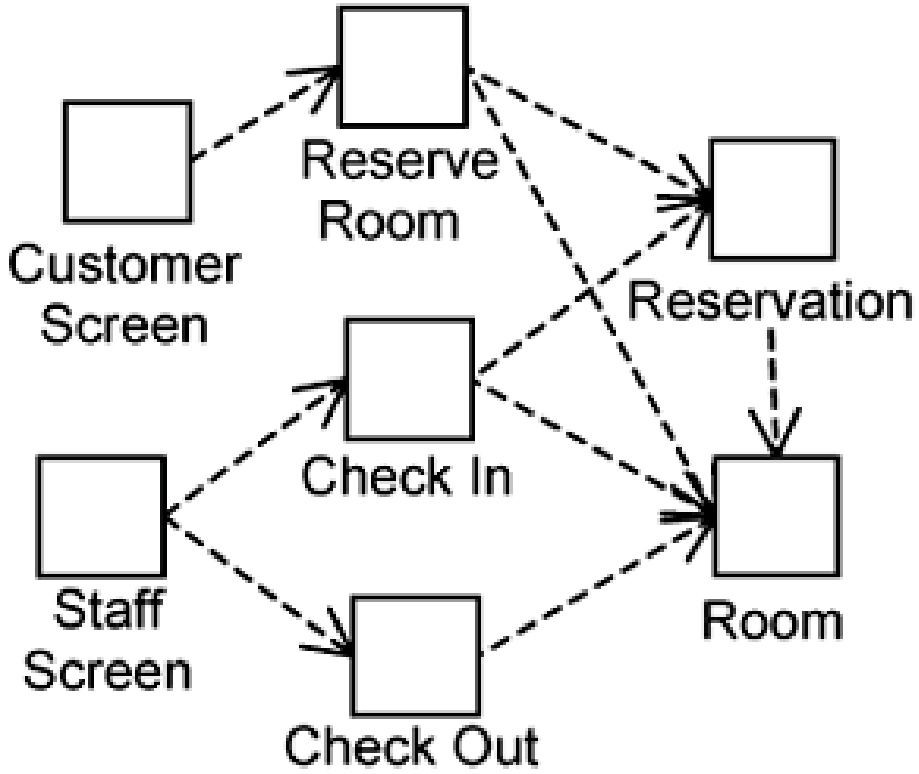
- *Aspect Weavers* کد منبع را پردازش نموده و جنبه‌ها را در برنامه و در *Pointcut* خاصی قرار می‌دهد
- سه روش برای قرار دادن جنبه‌ها
  - *Source Code pre-processing*
  - *Link-time weaving*
  - *Dynamic, Execution-time weaving*

# Aspect weaving



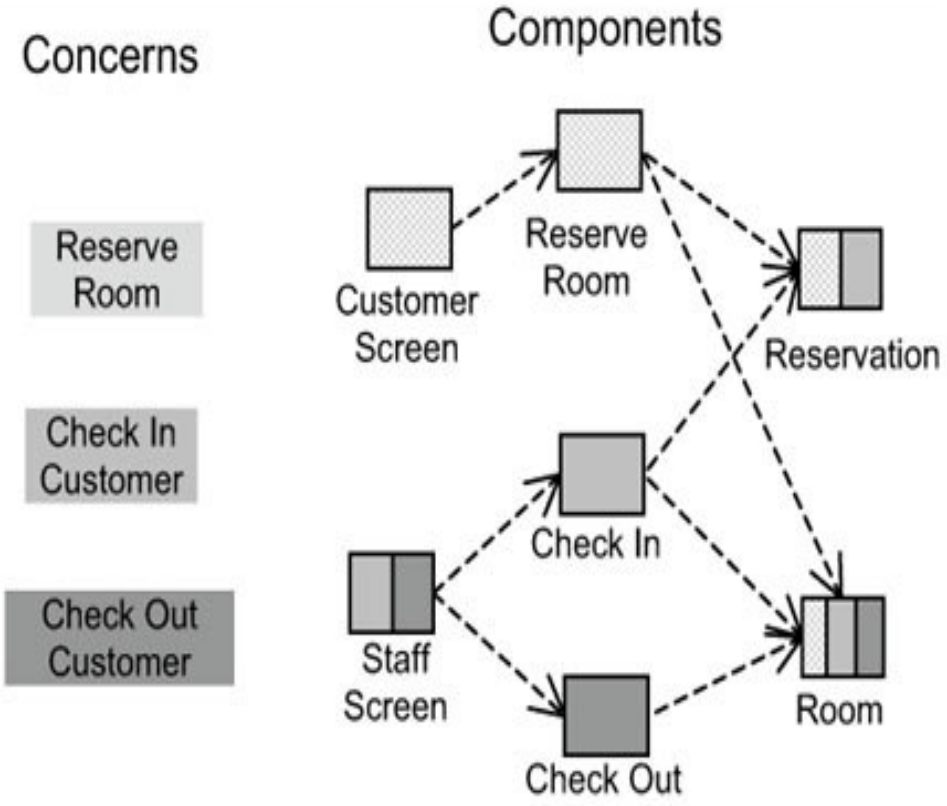
# نمونه استفاده از جنبه

- سیستم مدیریت هتل



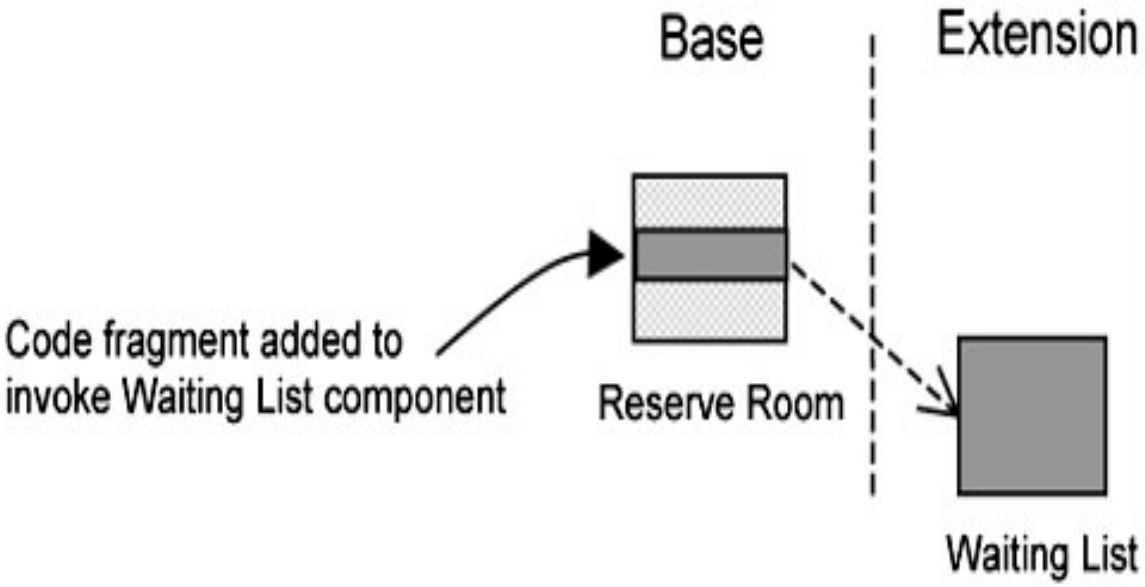
# نمونه استفاده از جنبه (ادامه)

- تحقق بخشیدن به دغدغه‌ها



# نمونه استفاده از جنبه (ادامه)

- تحقق بخشیدن به دغدغه‌ها



# نمونه استفاده از جنبه (ادامه)

- تحقق بخشیدن به دغدغه‌ها

| | Room | Reservation | Payment |
|---|---|---|---|
| Reserve Room | checkAvailability() | create() | |
| Check In Customer | assignCustomer() | consume() | createBill() |
| Check Out Customer | removeCustomer() | | payBill() |

# نمونه استفاده از جنبه (ادامه)

- کد پیاده‌سازی

```
1.    public aspect CheckInCustomer {
2.    …
3.    public void Room.assignCustomer ()
4.    {
5.    // code to check in customer
6.    }
7.    public void Reservation.consume()
8.    {
9.    // code to consume reservation
10.   }
11.   public void Payment.createBill()
12.   {
13.   // code to generate an initial outstanding bill
14.   }
```

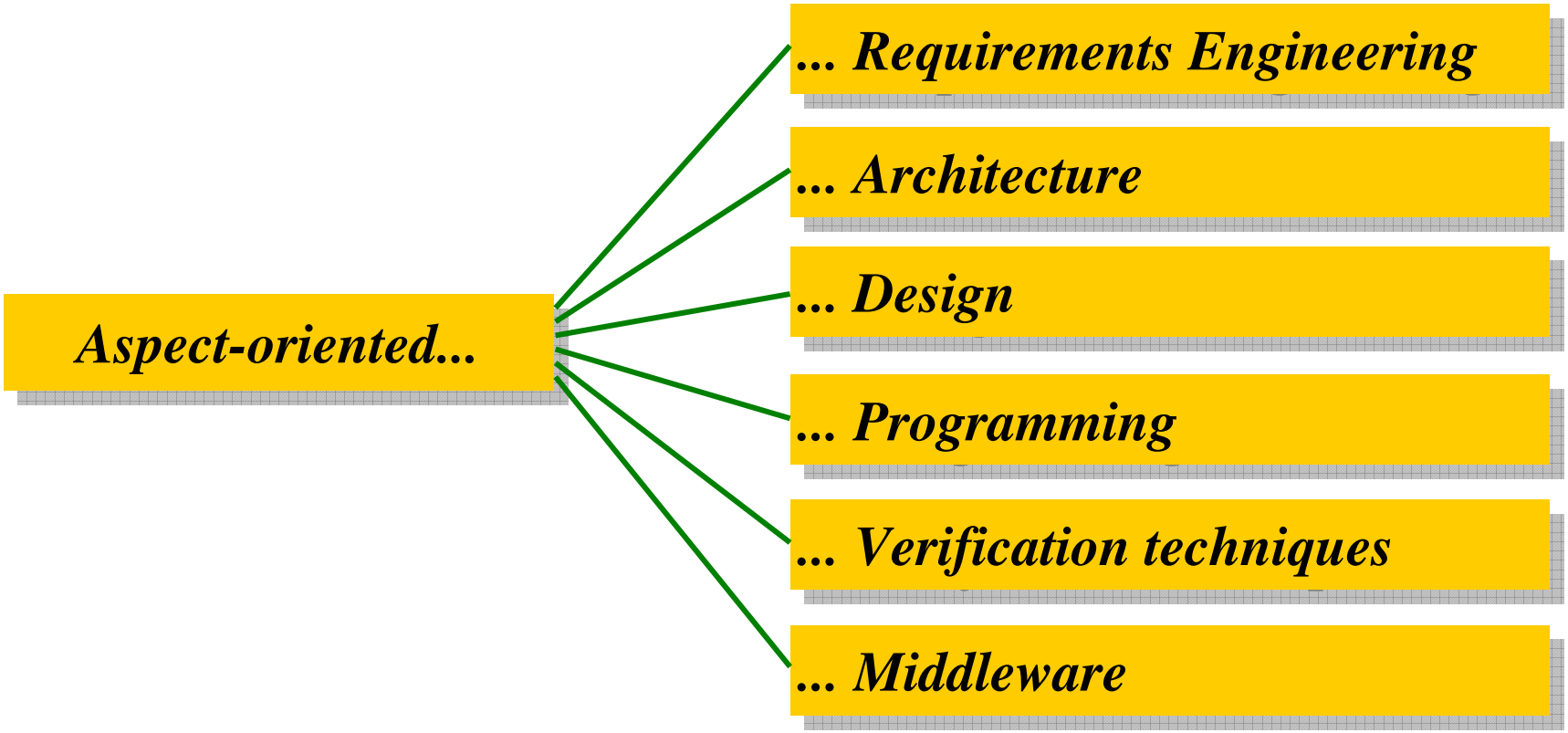
# شی و جنبه

- شباهت
  - هر دو دارای نوع هستند
  - می‌توانند به کلاس‌های دیگر یا جنبه‌های دیگر توسعه داده شوند
  - می‌توانند به‌صورت تجرید باشند
  - می‌توانند دارای فیلد، متدها و نوع داده باشند
- تفاوت
  - جنبه‌ها می‌توانند به صفات دیگر اعضا دسترسی داشته باشند

# توسعه نرم‌افزار مبتنی بر جنبه

- توسعه مبتنی بر جنبه به استفاده از جنبــه در تولیــد و توســعه نرم‌افزار اشاره دارد
- همانند متدولوژی‌های دیگر، متدولوژی‌های مبتنی بر جنبه نیــز فازهای مشابه خود را دارند، اما خود را مبتنی بر جنبه می‌دانند

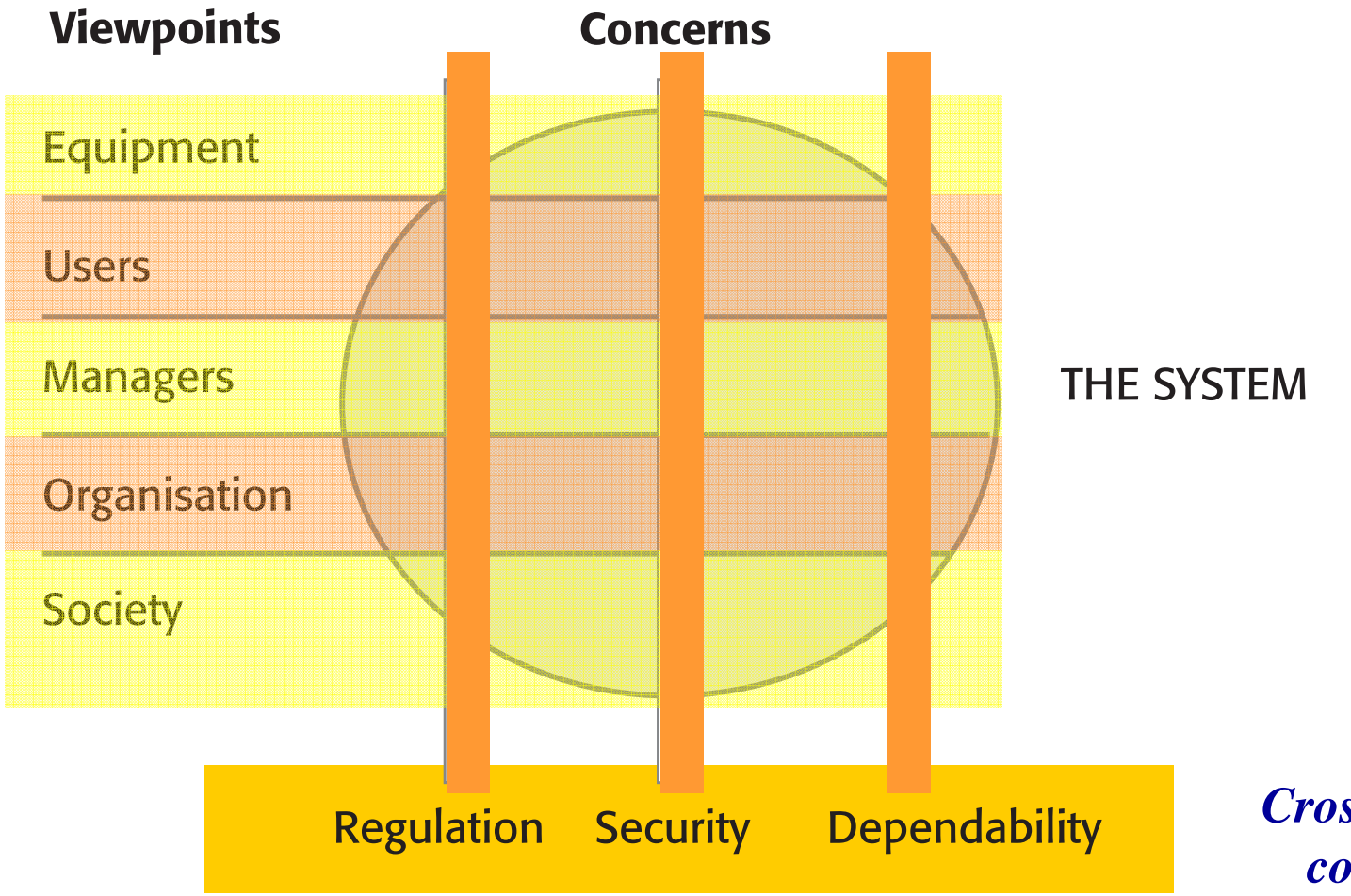
# توسعه نرم‌افزار مبتنی بر جنبه (ادامه)



# مهندسی نیازمندی‌های جنبه‌گرا

- روشی به‌منظور تمرکز بر روی دغدغه‌های مشتری که با توسعه جنبه‌گرا سازگار است
- پایه این روش *Viewpoint*ها هستند
  - استفاده از *Viewpoint*ها برای جداسازی دغدغه‌ها
  - استفاده از *Viewpoint*ها برای نمایش نیازمندی‌های گروه‌های مرتبط ذینفعان
  - *Cross-cutting concerns* دغدغه‌هایی که توسط تمام *Viewpoint*ها مشخص می‌شوند

# مهندسی نیازمندی‌های جنبه‌گرا (ادامه)



# مهندسی نیازمندی‌های جنبه‌گرا (ادامه)

- نیازمندی‌های اصلی (*Core Requirements*)
  - همان دغدغه‌های اصلی (*Core Concerns*) هستند
  - هدف از توسعه سیستم دستیابی به نیازمندی‌های اصلی است
- نیازمندی‌های توسعه‌ای (*Extension Requirements*)
  - مجموعه‌ای از *Cross-cutting Concerns* هستند

# مهندسی نیازمندی‌های جنبه‌گرا (ادامه)

- نمونه: فروشگاه زنجیره‌ای آنلاین
  - مجموعه‌ای از فروشگاه‌ها که خدمات خود را به صورت اینترنتی ارائه می‌دهند
  - کاربران با مشاهده کالاها، درخواست خرید خود را ارائه می‌دهند
  - کالا برای کاربر بوسیله پست ارسال می‌شود
  - این فروشگاه‌ها دارای یک انبار مرکزی هستند که کالاهای خود را از آنجا تهیه و در صورت خرابی به آنجا ارسال می‌نمایند
  - در صورت خرابی یا نقص در کالا، کاربر می‌تواند ۴۸ ساعت پس از خرید آن را عودت دهد

# مهندسی نیازمندی‌های جنبه‌گرا (ادامه)

- نمونه دیدگاه: فروشگاه زنجیره‌ای آنلاین
  - دید کاربران
    - جستجوی کالاها
    - مشاهده موجودی هر کالا
    - مشاهده موجودی کالا در فروشگاه‌های خاص
    - پیدا کردن نزدیکترین فروشگاه
    - درخواست خرید کالا
    - گزارش عدم دریافت کالا
    - گزارش خرابی کالای خریداری شده
    - ...

# مهندسی نیازمندی‌های جنبه‌گرا (ادامه)

- نمونه دیدگاه: فروشگاه زنجیره‌ای آنلاین
  - دید فروشنده
    - تقاضای کالا از انبار مرکزی
    - جستجوی کالاها
    - مشاهده موجودی کالاها
    - مشاهده، بررسی و تائید درخواست خرید کالا
    - مشاهده، بررسی و تائید گزارش خرابی کالا
    - ارسال کالای دارای نقص به انبار مرکزی
    - ...

# مهندسی نیازمندی‌های جنبه‌گرا (ادامه)

- نمونه دیدگاه: فروشگاه زنجیره‌ای آنلاین
  - دید انباردار
    - ارسال کالا به فروشگاه متقاضی
    - جستجوی کالاها در انبار مرکزی
    - بررسی و تعویض کالای دارای نقص
    - بارکدگذاری کالاهای انبار
    - بروز نگهداری اطلاعات موجودی انبار
    - ...

# مهندسی نیازمندی‌های جنبه‌گرا (ادامه)

- نمونه دیدگاه: فروشگاه زنجیره‌ای آنلاین
  - نیازمندی‌های اصلی
    - کاربران می‌توانند کالاها را جستجو و مشاهده نمایند
    - کاربران مجاز (شناسایی شده) می‌توانند اطلاعات کامل کالا را مشاهده و درخواست خرید کنند
    - فروشندگان می‌توانند موجودی کالای خود را مشاهده و تغییر دهند
    - انباردار می‌بایست برنامه زمانبندی تعویض کالا را ایجاد نماید
    - ...

# مهندسی نیازمندی‌های جنبه‌گرا (ادامه)

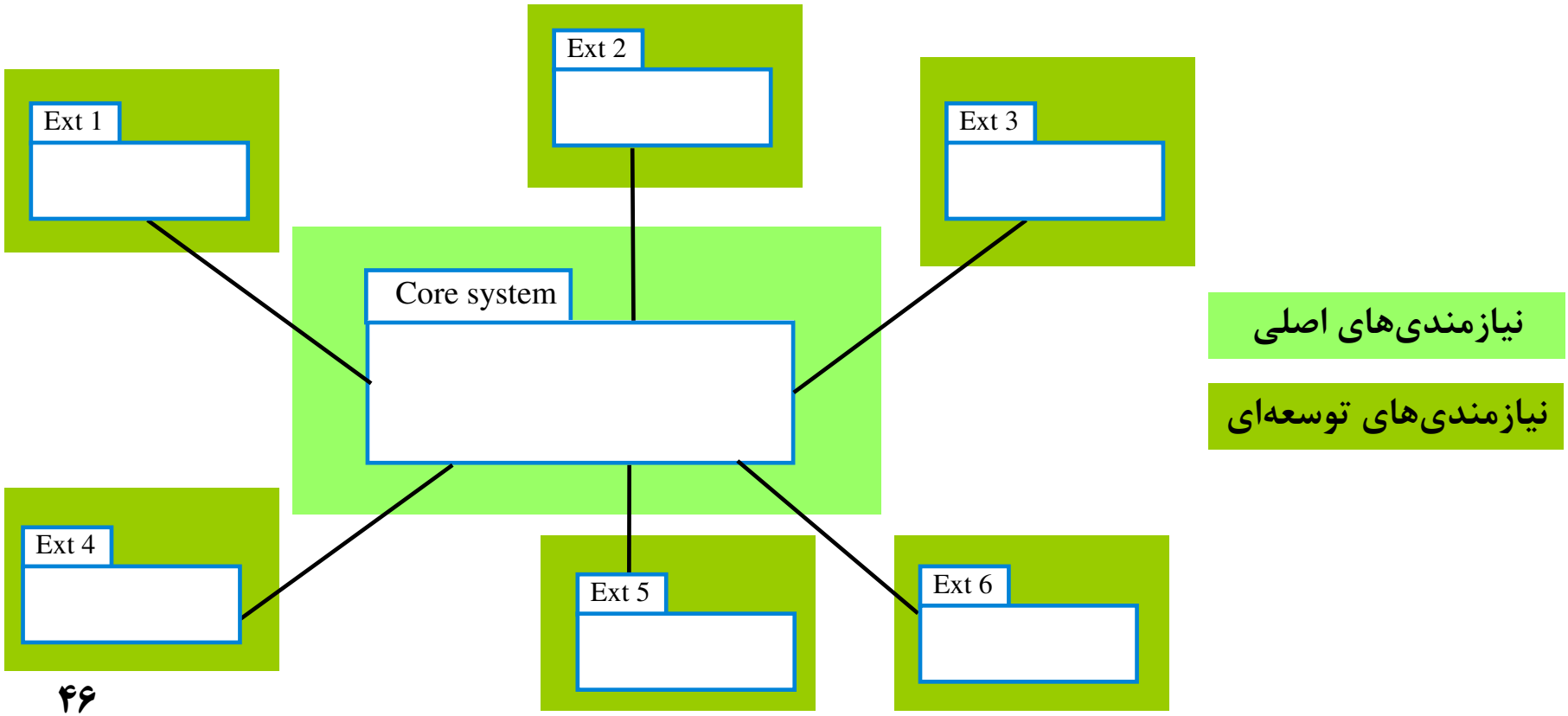
- نمونه دیدگاه: فروشگاه زنجیره‌ای آنلاین
  - نیازمندی‌های توسعه‌ای

غیروظیفه‌مندی وظیفه‌مندی

- کاربر و فروشنده می‌توانند برنامه زمانی تعویض کالا را مشاهده نماید (دید کاربر + دید فروشنده + دید انباردار)
- کاربر و انباردار می‌توانند گزارش خرابی کالای تائید شده توسط فروشنده را مشاهده نمایند (دید کاربر + دید فروشنده + دید انباردار)
- اطلاعات کاربران، فروشندگان و انبارداران به صورت رمزشده ذخیره می‌شود

# معماری جنبه‌گرا

- معماری سیستم جنبه‌گرا برپایهٔ معماری سیستم‌های اصلی (*Core Systems*) و توسعه‌ها (*Extensions*) است



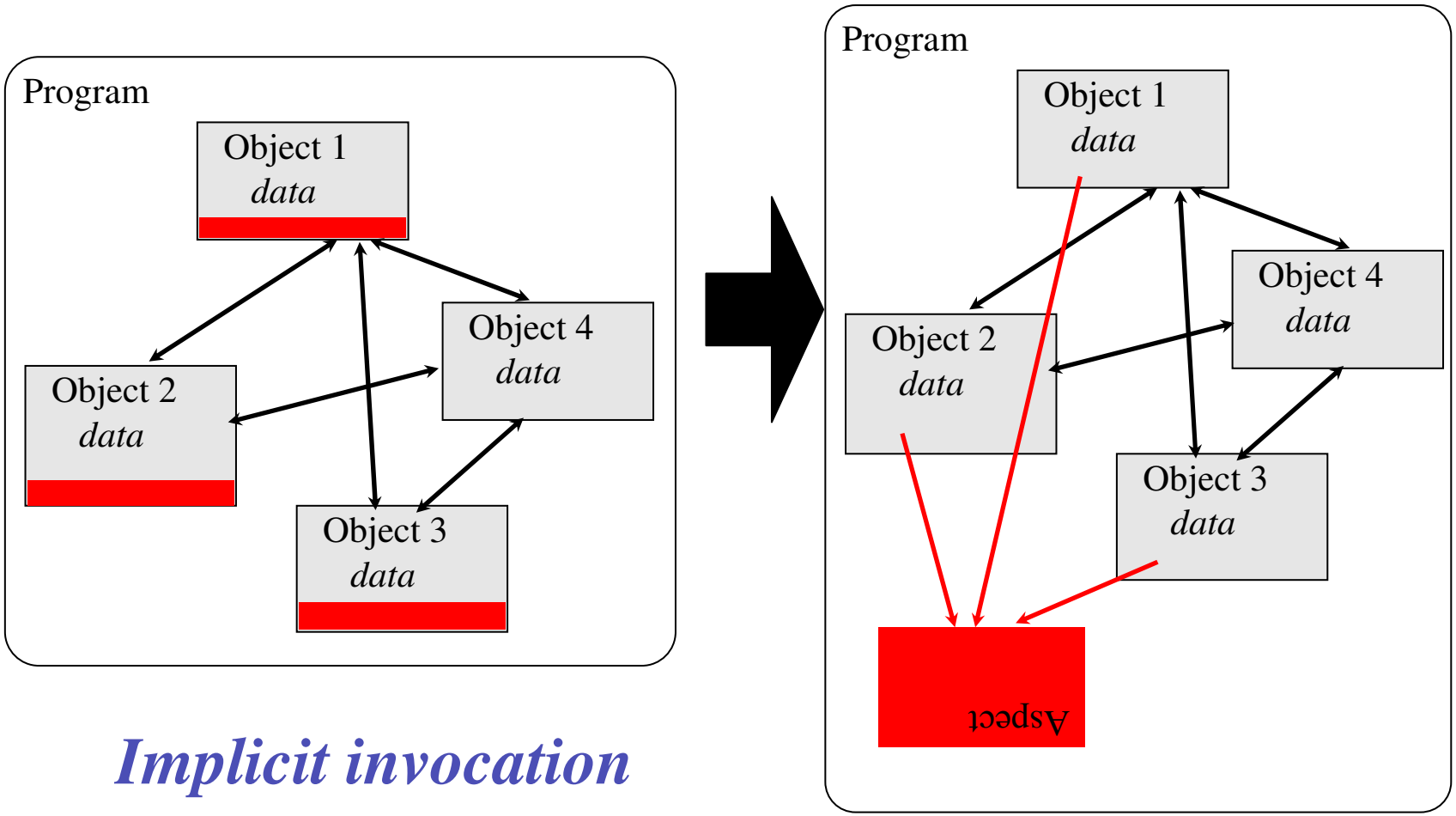
# معماری جنبه‌گرا (ادامه)

- انواع توسعه‌ها
  - توسعه‌های کارکردی ثانویه
    - قابلیت‌هایی کارکردی اضافه‌ای را به سیستم اصلی می‌افزاید
  - توسعه‌های سیاستی (*Policy*)
    - قابلیت‌های کارکردی را برای حمایت از سیاست‌های سازمانی (مانند امنیت) می‌افزاید
  - توسعه‌های کیفیت سرویس (*QoS*)
    - قابلیت‌های کارکردی را برای کمک به نگهداری کیفیت سرویس می‌افزاید
  - توسعه‌های زیرساختی (*Infrastructure*)
    - قابلیت‌های کارکردی را برای حمایت از پیاده‌سازی سیستم روی سکوهای مختلف می‌افزاید

# طراحی جنبه‌گرا

- فرآیند طراحی سیستمی از جنبه‌ها برای پیاده‌سازی *cross-cutting concerns* و دیگر نیازمندی‌هایی که در طول فرآیند مهندسی نیازمندی‌ها شناسایی شده‌اند، استفاده می‌کند
- طراحی جنبه‌گرا باید به‌گونه‌ای انجام شود که دو مشکل *Scattering* و *Tangling* به کمترین حد خود برسد

# طراحی جنبه‌گرا (ادامه)



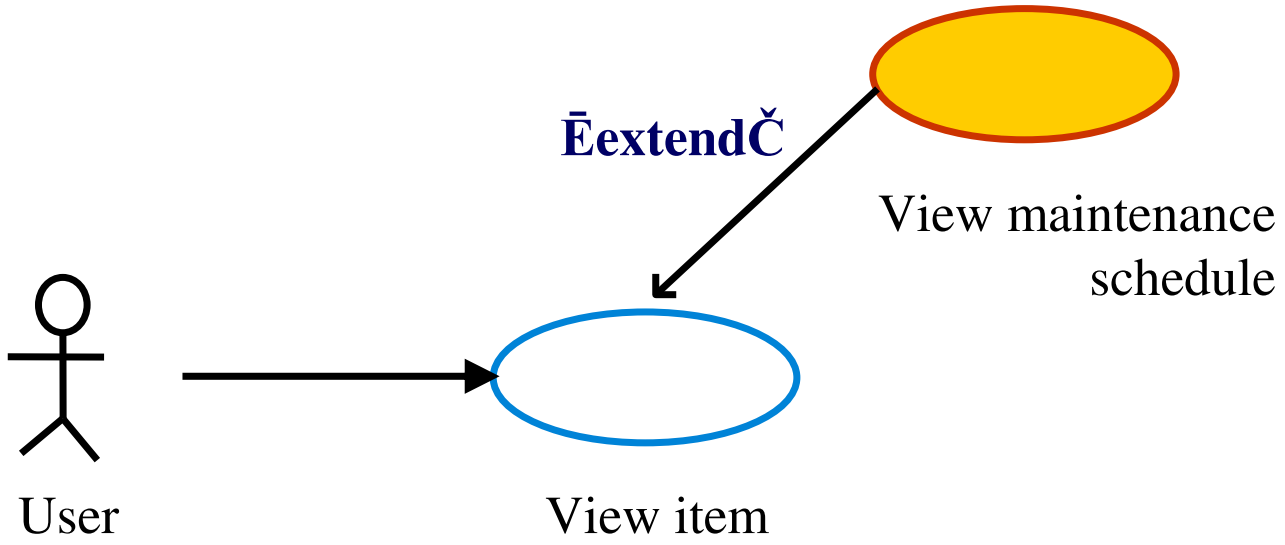

*Implicit invocation*

# طراحی جنبه‌گرا (ادامه)

- مورد کاربری می‌تواند به عنوان جنبه در نظر گرفته شود
- ایده استفاده از مورد کاربری برای جنبه توسط *Jacobsen* و با استفاده از مفاهیم جدیدی چون *use-case slices* و *use-case modules* ارائه شده است
- مورد کاربری توسعه‌ای برای نمایش مدل معماری جنبه‌گرای اصلی + توسعه‌ها بکار می‌رود

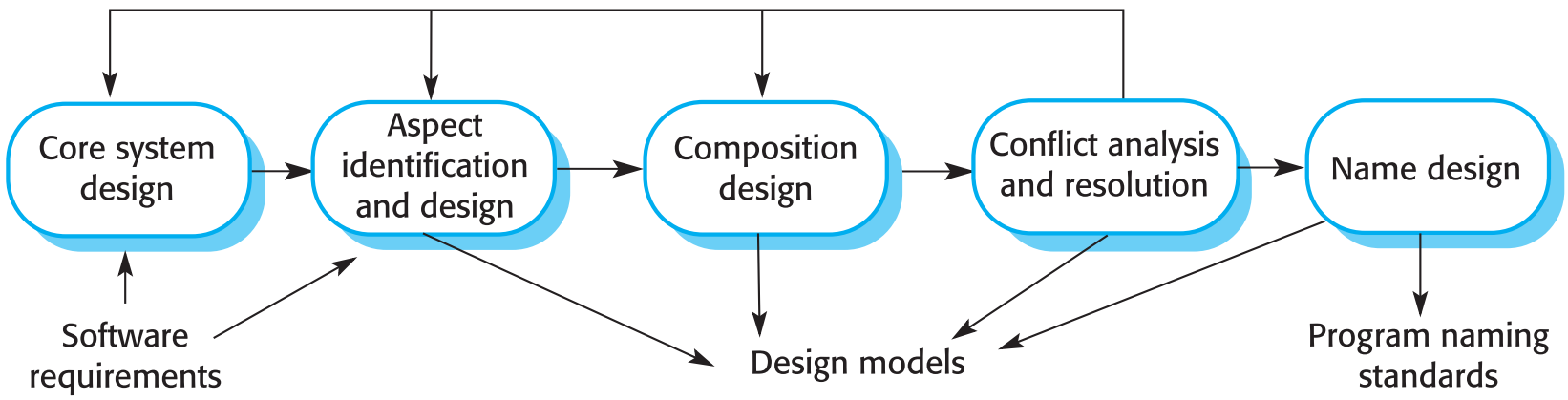
# طراحی جنبه‌گرا (ادامه)

- مورد کاربری توسعه‌ای



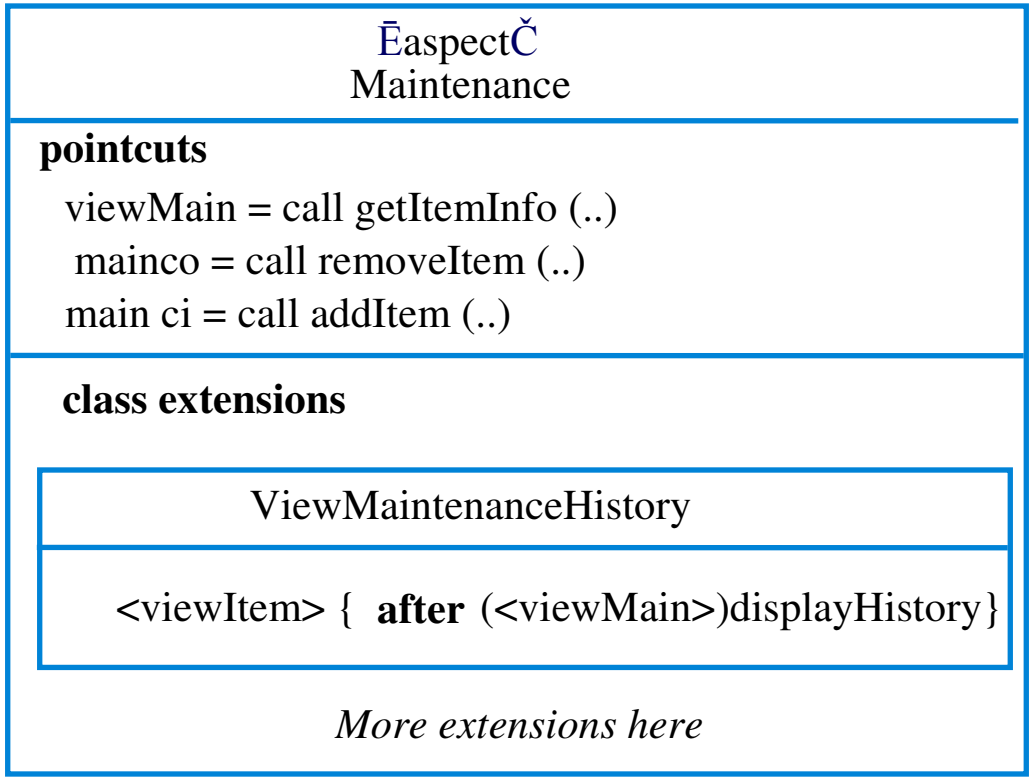
# طراحی جنبه‌گرا (ادامه)

- فرآیند انجام طراحی جنبه‌گرا
  - برای بررسی عدم همپوشانی جنبه‌ها باید تعارض بین جنبه‌ها شناسایی و رفع شود



# طراحی جنبه‌گرا (ادامه)

- استفاده از UML برای جنبه‌ها
  - مدلی نه چندان کامل از تعریف جنبه



# برنامه‌نویسی جنبه‌گرا

- انقلابی در شیوه فکر کردن در زمینه مهندسی نرم‌افزار
- برای اولین بار توسط Gregor Kiczales در سال ۱۹۹۷ معرفی شد
- مکانیزم جداسازی پیشرفته دغدغه‌ها نیز اطلاق می‌شود
- زبان‌های پشتیبانی‌کننده جنبه
  - AspectJ: یک بسته که امکانات AOP را به زبان برنامه نویسی جاوا اضافه می‌کند
  - Aspeckwerkz , Jboss-AOP , HyperJ

# آزمایش جنبه‌گرا

- همانند سایر سیستم‌ها، سیستم‌های جنبه‌گرا می‌توانند به‌صورت **جعبه سیاه** مورد آزمایش قرار گیرند
- آزمایش **جعبه سفید** آنها پیچیده و وابسته به کد برنامه
- مشکلات آزمایش جنبه‌ها
  - رابط جنبه چگونه می‌تواند آزمایش شود؟
  - چگونه جنبه‌ها می‌توانند مستقل از سیستم اصلی آزمایش شوند؟
  - چگونه می‌توان آزمایشی طراحی نمود که همه نقاط اتصال اجرا شده و جنبه‌های مناسب مورد آزمایش قرار گیرند؟

# آزمایش جنبه‌گرا (ادامه)

- تولید گراف جریان برنامه برای جنبه‌ها غیرممکن است، به همین دلیل طراحی آزمایشاتی که تضمین دهند همه ترکیبات جنبه‌ها و کد اصلی اجرا شوند، دشوار است
- پوشش آزمایش (*test coverage*) معنای خاصی ندارد
  - کد هر جنبه یکبار اجرا شود؟
  - کد هر جنبه در هر نقطه اتصالی که جنبه قرار داده شده است یکبار اجرا شود؟
  - ؟