Aspect Oriented Software Engineering (AOSE)

Chapter 13

Objectives

After this lecture, you will:

- Understand why the separation of concerns is a good guiding principle for software development
- Understand the fundamental ideas underlying aspects and aspect-oriented software development
- Understand how an aspect-oriented approach may be used for requirements engineering, software design, and programming
- Be aware of the difficulties of testing aspect-oriented systems.

The Separation of Concerns

Separation of concerns:

Organize software so that each element in the program (class, method, procedure, etc.) does one thing and one thing only.

Concern:

Something that is of interest or significance to a stakeholder or a group of stakeholders.

Different Types of Stakeholder Concern

Functional Concerns

- Functionality
- E.g. in a train control system – train braking is a functional concern

Different Types of Stakeholder Concern

Quality of Service Concerns

- Quality
- E.g. performance, reliability

Different Types of Stakeholder Concern

Policy Concerns

- Policy that governs the use of a system
- E.g. security, safety, business rules

Different Types of Stakeholder Concern

System Concerns

- Related to the attributes of the system as a whole.
- E.g. maintainability, configurability

Different Types of Stakeholder Concern

Organizational Concerns

- Organizational goals & priorities.
- E.g. produce SW within budget, make use of existing assets, maintain reputation

Cross Cutting Concerns

Cross-Cutting Concerns

Core Concerns:

- Those functional concerns that relate to its primary purpose
- E.g., a hospital patient information system: core functional concerns are the creating, editing, retrieval, and management of patient records.

Cross-Cutting Concerns

Secondary Functional Concerns:

- Non-functional requirements

Cross-Cutting Concerns

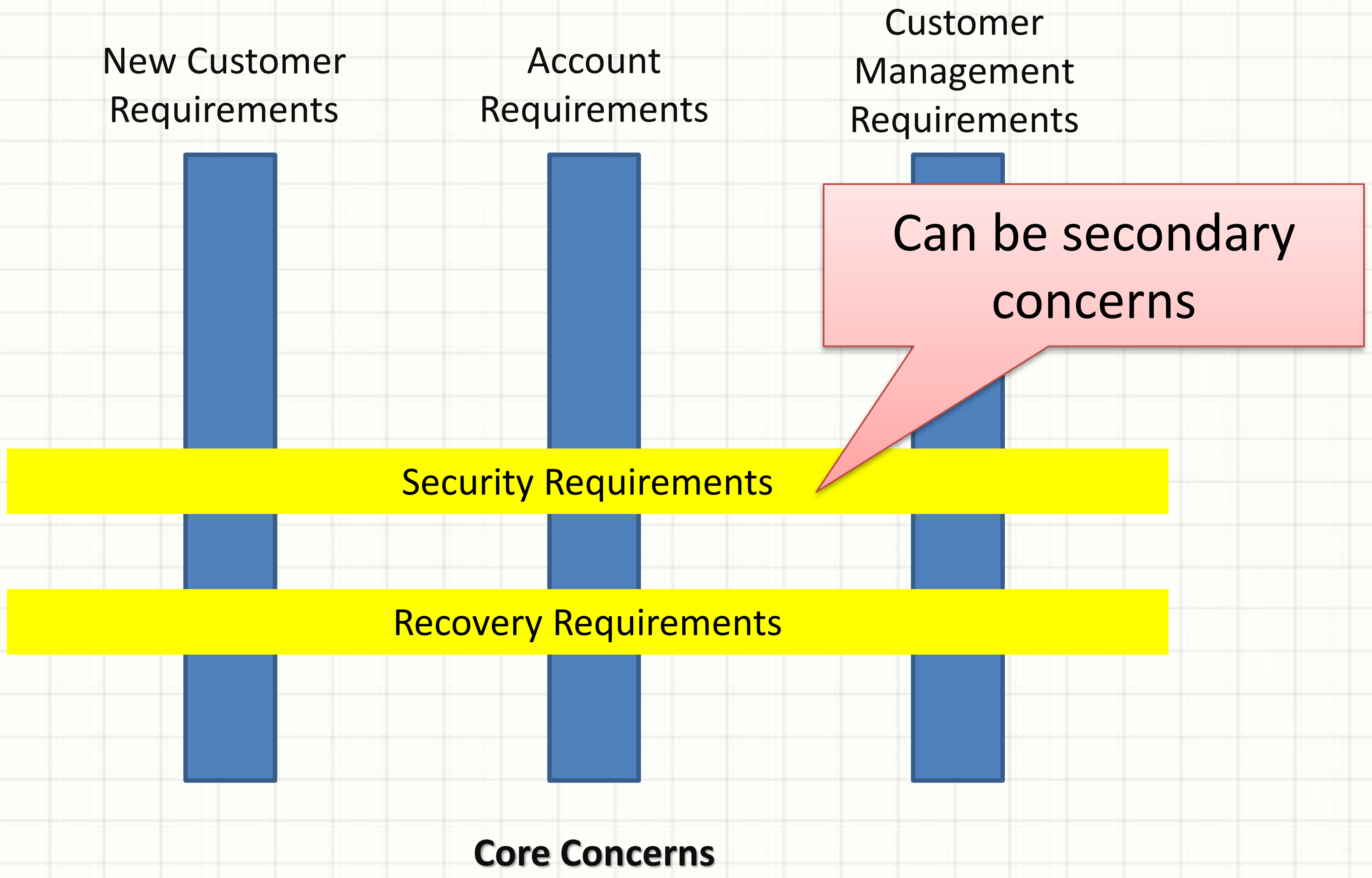

New Customer Requirements
Account Requirements
Customer Management Requirements
Can be secondary concerns
Security Requirements
Recovery Requirements
Core Concerns

Figure 21.1 pg 569 cross-cutting concerns Internet banking service.

AOSE Basic Concepts

- Concern
- Separation of Concern
- Different types of Concern
- Core vs. Secondary Concern
- Cross-cutting concern

Exercise

You are a software manager who propose to use Aspect Oriented Software Engineering (AOSE) to develop an online movie ticketing system.

Construct a simple diagram to show THREE core concerns and ONE cross-cutting concern.

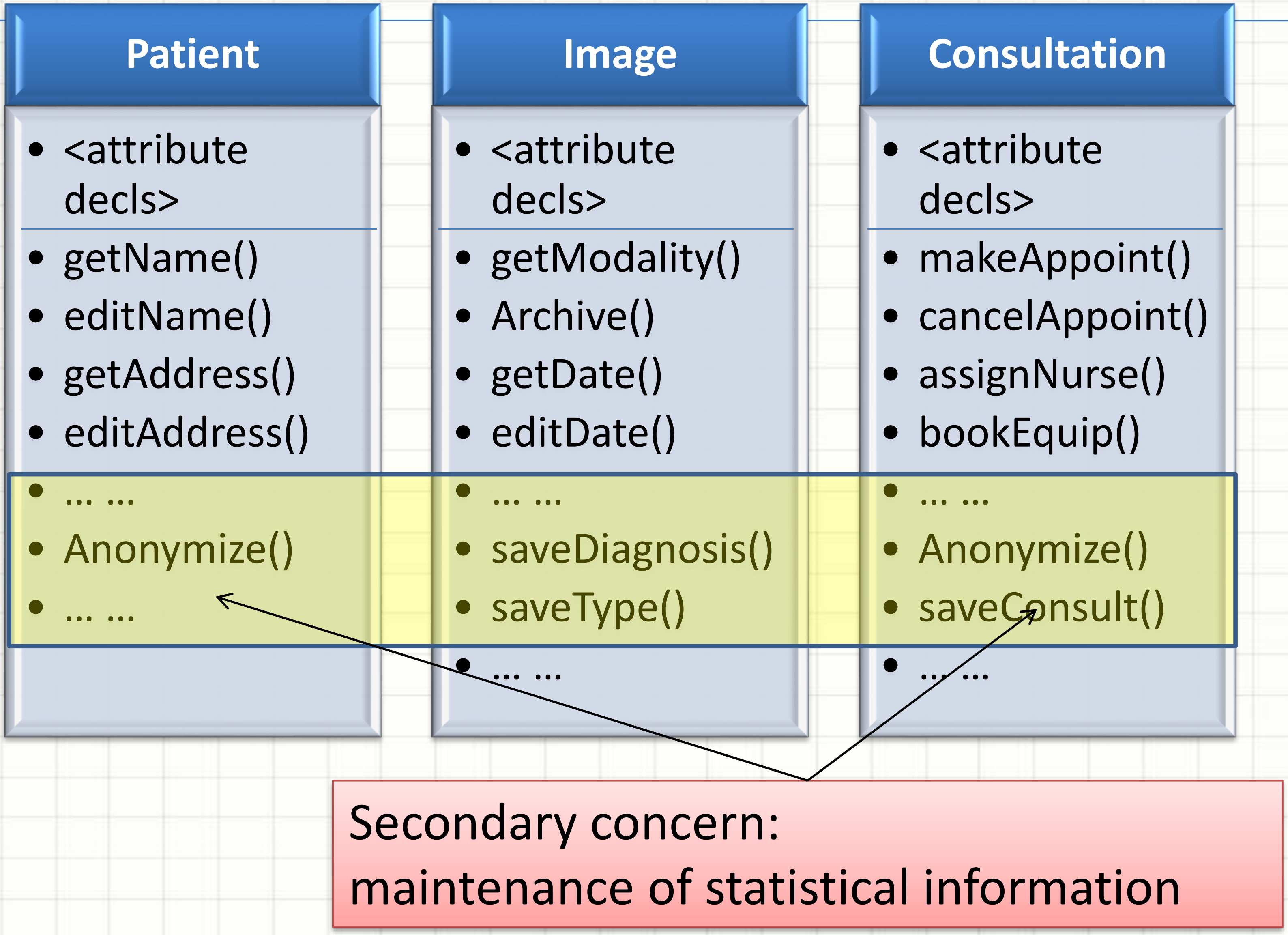
Problem with Programming Language Abstraction

Tangling

Secondary Concern:
Synchronization Concern

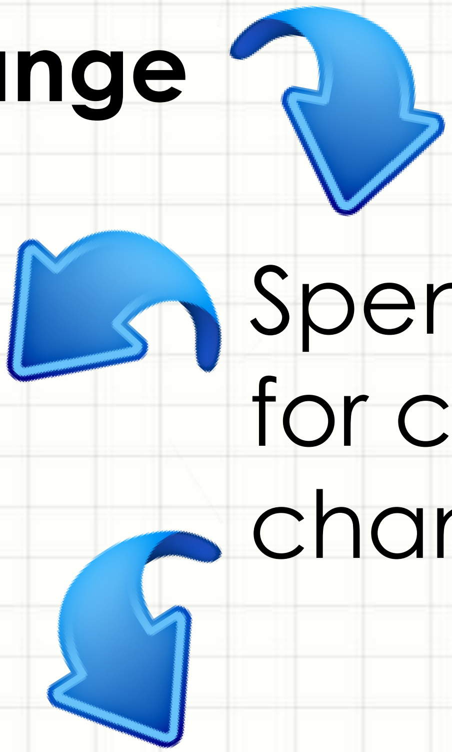
```
Synchronized void put (SensorRecord rec)
{
        //check that there is space in the buffer; wait if not
        if (numberOfEntries == bufsize)
                wait();
        //add record at end of buffer
        store[back] = new SensorRecord(rec.sensorid, rec.sensorVal);
        back = back + 1;
        //if at end of buffer, next entry is at the beginning
        … …
        //indicate that buffer is available
        notify();
}//put
```

Scattering


Patient
<attribute decls>
getName()
editName()
getAddress()
editAddress()
… …
Anonymize()
… …
Image
<attribute decls>
getModality()
Archive()
getDate()
editDate()
… …
saveDiagnosis()
saveType()
… …
Consultation
<attribute decls>
makeAppoint()
cancelAppoint()
assignNurse()
bookEquip()
… …
Anonymize()
saveConsult()
… …
Secondary concern:
maintenance of statistical information

Problems with Tangling & Scattering

SRS change



Spend time looking for components to change

Risky

Expensive

Aspects, Join Points, and Pointcuts

Aspects, Join Points, and Pointcuts

Advice	- Code
Aspect	- Define concern, pointcut & advice associate with concern
Join Point	- Event
Pointcut	- Statement defines join point

Aspects, Join Points, and Pointcuts

Join Point Model	- A set of events
Weaving	- Incorporation of advice code at the specified join points by an aspect weaver

Aspects, Join Points, and Pointcuts

```
aspect authentication
{
    before: call(public void update*(…)//this is a pointcut
    {
        //this is the advice that should be executed when woven into executing sys
        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
```

Aspects, Join Points, and Pointcuts

Join Points Model example

- Call Event
- Execution Event
- Initialization Event
- Data Event
- Exception Event

$\Bigg\}$ Advice can be woven into the join points

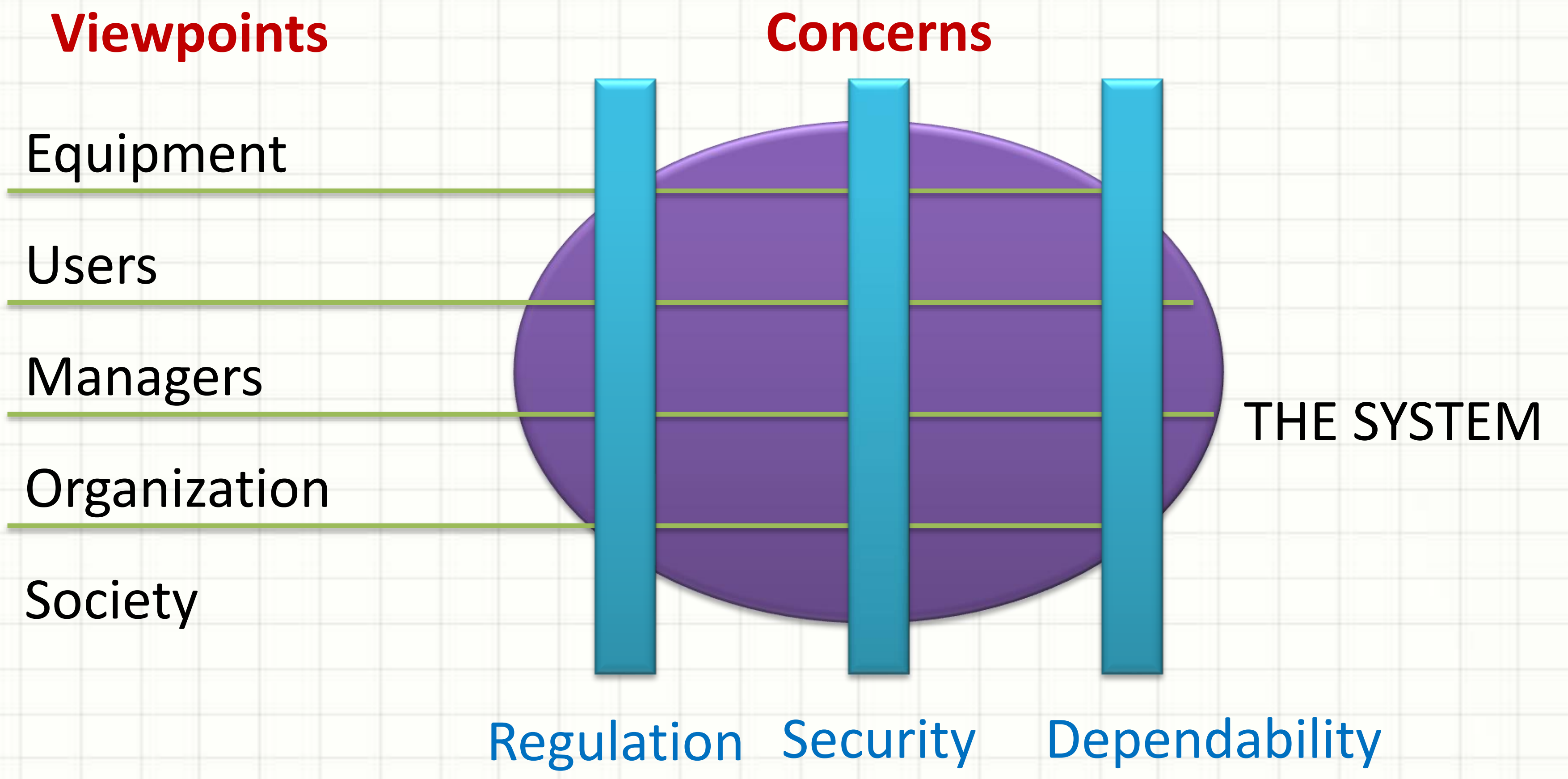
Aspects, Join Points, and Pointcuts

Aspect Woven

- responsible to include the advice at the join points specified in the pointcuts

Software Engineering with Aspects

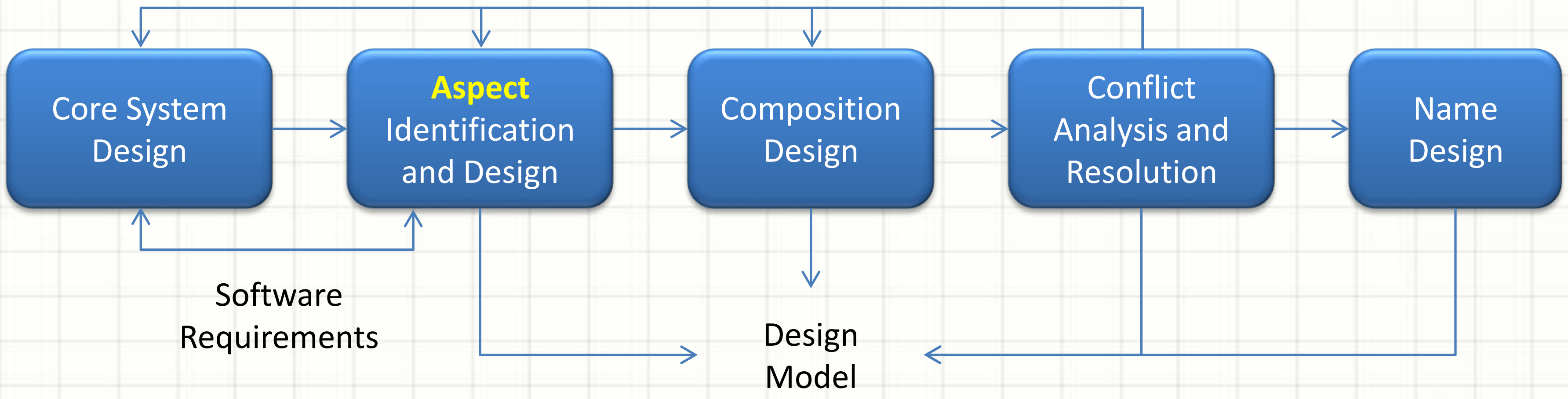
Concern-oriented Requirements Engineering



Aspect-oriented Design & Programming

Design aspects based on concerns

A Generic Aspect-oriented Design Process



Verification & Validation

Verification & Validation

- Program inspections Code Reading Tool
- White-box testing Code → Test?
- How should aspects be specified so that tests for these aspects may be derived?

Test coverage? Test plan?

Verification & Validation

- How can aspects be tested independently of the base system with which they should be woven?
- How can aspect interference be tested?
- How can tests be designed so that all program join points are executed and appropriate aspect tests applied?