Design Principles Behind PATAGONIA

Hernán Wilkinson
hernan.wilkinson@10pines.com

10 Pines | agile software development & services

What is Patagonia?

- A beautiful place…

10 Pines

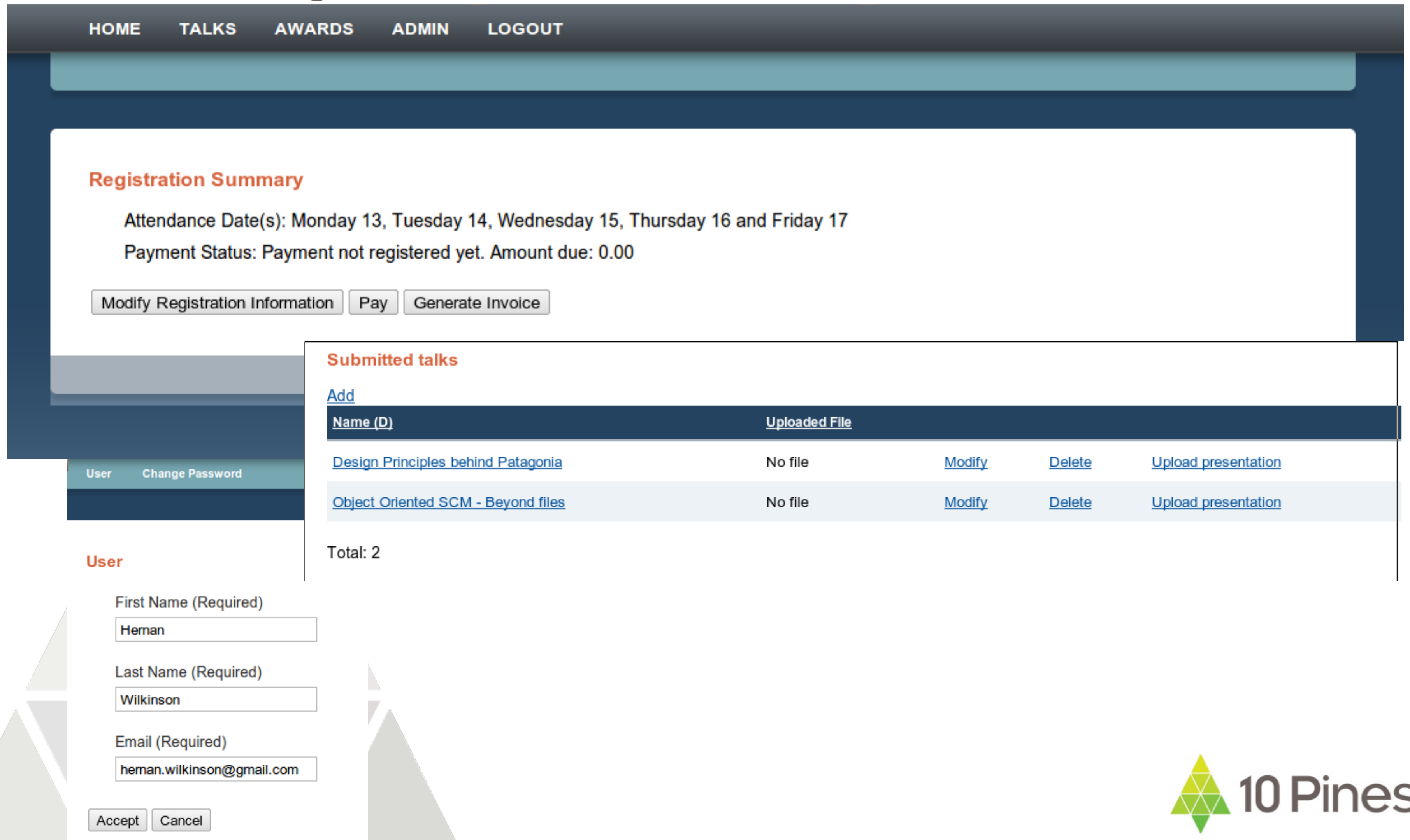
But also…

- Conference Registration System
- Sponsored by ESUG
- Developed by 10Pines
- Developed with:
  - Pharo
  - Seaside 3.0 (using JQuery in some places)
- Production:
  - Pharo
  - Amazon Cloud
  - File based persistence (ReferenceStream… but GemStone is the best option, no doubt about it ☺)

10 Pines

What is Patagonia?

- User Registration front end



What is Patagonia?

- Group Registration front end

HOME GROUP MEMBERS ADMIN LOGOUT

Group Manager Short Description

Organization name: 10Pines

Payment Status: No status available. The group has no members

Modify Affiliation | Register Attendee | Pay | Generate Invoice

Hernan Wilkinson / Saturday, September 4, 2010 - Platform: Pharo - Server: ip-10-224-107-3.eu-west-1.compute.internal

Attendees

Register Attendee Register Myself as Attendee

Name (D)	Email	Has Paid?

Total: 0

What is Patagonia?

- Administration front end

HOME CONFERENCE REGISTRATION STATISTICS ADMIN LOGOUT

Welcome to the Patagonia System Administration

Hernan Wilkinson / Saturday, September 4, 2010 - Platform: Pharo - Server: ip-10-224-107-3.eu-west-1.compute.internal

Conference Configuration

Name

ESUG 2010

Description

International Smalltalk Conference

Dates

From

Day 13 Month 9 Year 2010

To

Day 17 Month 9 Year 2010

Attendees

Register Attendee with New User Register Attendee with Existing User Register Myself as Attendee Export

Name (D)	Email	Country	Affiliation	Has Paid?	Checked?				
Adriaan		Netherlands	Delta Lloyd	Yes (Change)	No (Change)	Modify	Delete	Modify User	Reset Password
Adrian		Switzerland	Cmsbox & netstyle.ch	Yes (Change)	No (Change)	Modify	Delete	Modify User	Reset Password

10 Pines

Some Statistics…

- Programming Errors: 1! (Fixed already)
- Functional gaps: some… we had to adjust as always
- Tests: 136 (no as many as I would like…)
- Classes: 269 (It includes some not functionality packages)
- Methods: 2415
- Lines Of Code: 16297
- Average Lines/Method: 6 (including comments… should be less)
- To do's: 55 ☺

10 Pines

Some info…

- Used at ESUG 2010
- License: MIT
- Code: www.squeaksource.com/Patagonia
- Limitations:
  - Supports only paid conferences
  - Some functionality is coupled with ESUG
    - Invoice
    - Payment
  - Registration wizard not configurable
  - And others…

10 Pines

Why “Design Principles Behind…” ?

We will see some design issues…

and

Some tips about how to avoid them…

(sorry Dan for stealing your title…)

10 Pines

Some Important design problems we saw…

Abstractions created from a technical perspective, not from the domain problem one

- UserDAO
- UserService
- … and User is just a data structure

Essential - Accidental

Some Important design problems we saw…

Objects without clear responsibilities

Schizophrenic objects

- Too many responsibilities
- Representing too many entities

Essential - Accidental

10 Pines

Some Important design problems we saw…

Tons of "nil doesNotUnderstand:" (or NullPointerException in "that great" language ☺)

– Difficult to see when an object "is complete"

Essential - Accidental

10 Pines

Some Important design problems we saw…

Essential - Accidental

Constraints not part of the model

- Lots of invalid objects
- Models that "do not teach" new programmers

10 Pines

Design Tips' Goals

- Write Robust Software
- Write software that is easy to understand and therefore to maintain
  - The model has to provide means for the humans to understand it easily
  - The model has to help you learn it!
  - The model has to tell us when we make mistakes as soon as we make them (immediate feedback)
- Computers to do more and programmers less
  - We will not care if it takes more time
  - We will prefer human time vs. computer time

10 Pines

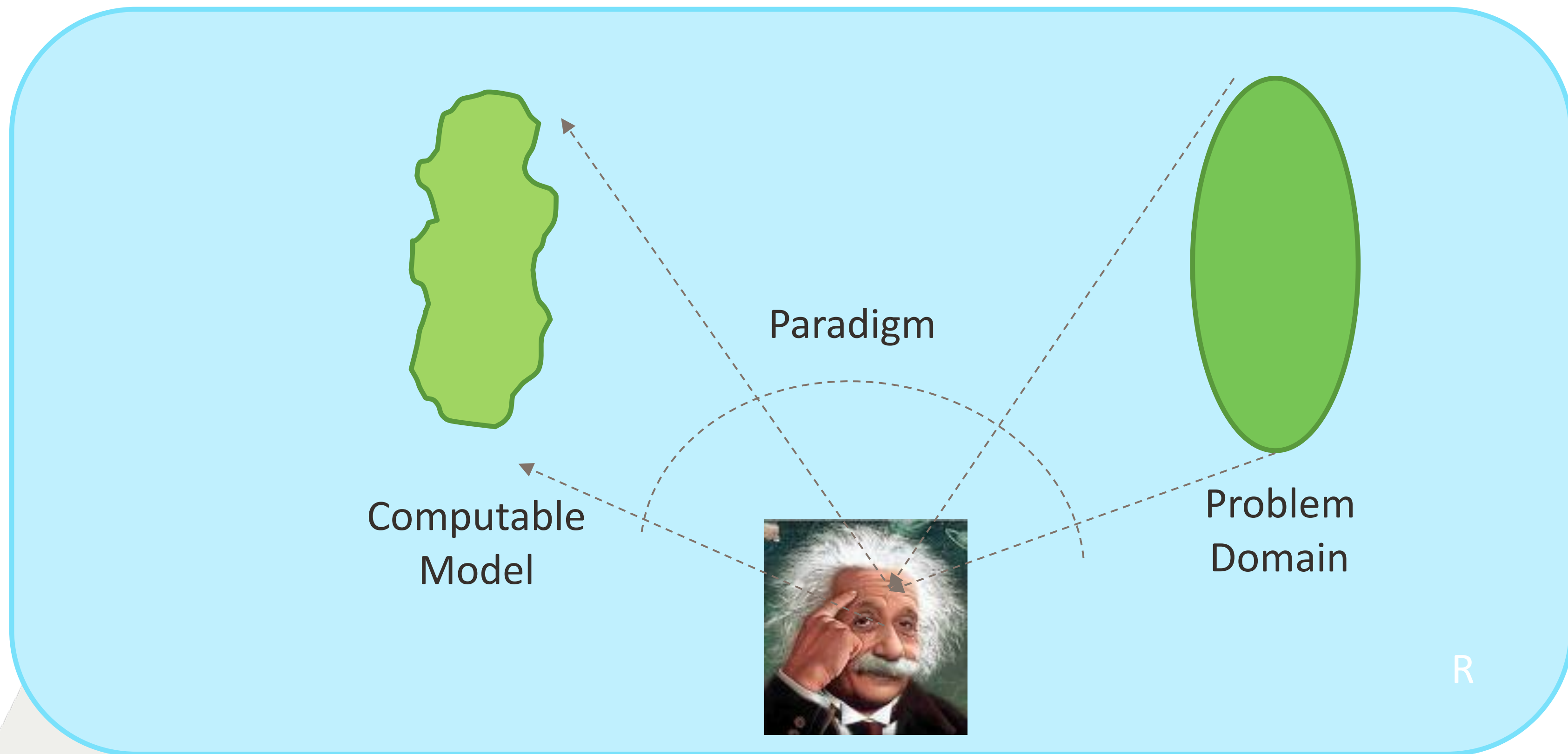
How do you know if they are good?

… well you don’t

- Based on some years of experience…
  - I started writing down these “tips” more that six years ago
  - I used them to develop different systems (financial/desktop, sales/web, patagonia, etc)
- Not sure if they apply to any kind of system

Let’s define software

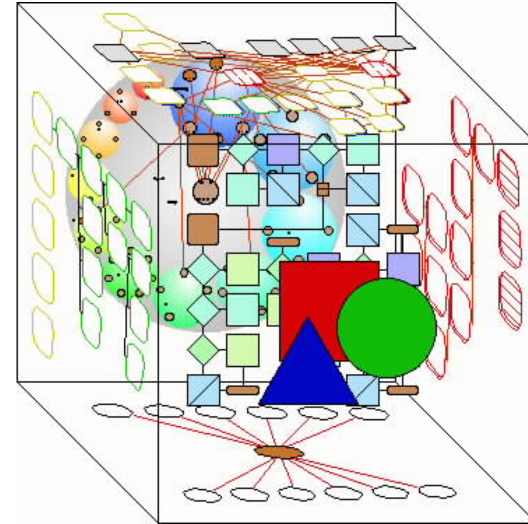
Computable **Model** of a Problem Domain



10 Pines

And software development as…

Knowledge acquisition



Knowledge representation

A learning process… ☺

10 Pines

Software development

Natural Language
- Informal
- Incomplete
- Tacit

A difficult task

Programming Language
- Formal
- Complete
- Explicit

Analysis

Progra mming

Design

Diagrams
- Informal
- Incomplete
- Explicit

10 Pines

Example – Patagonia Requirement

- We cannot have two conferences with the same name
- A talk can not be registered twice (they can not have the same name)
- We use the names as unique identifiers but…
  - Comparing them should not be case sensitive
  - Blanks don’t matter
  - They cannot be empty
- Where do we put that knowledge? Should we represent that knowledge?

10 Pines

Example

For most people, they are just Strings

For some solitary people (like me), they have enough behavior to be reified. I called them Name

10 Pines

Another Example

!∀#∃%&∋()*+,−./0123456
789:;<=>?≅ΑΒΧΔΕΦΓΗΙ
ϑΚΛΜΝΟΠΘΡΣΤΥςΩ
ΞΨΖ[∴]⊥_ ‾αβχδεφγηι
φκλμνοπθρστυϖωξψζ{
|}~€ϒ′≤/∞ƒ♣♦♥♠↔←↑
→↓°±″≥×∝∂•÷≠≡≈…|—
↵ℵℑℜ℘⊗⊕∅∩∪⊃⊇⊄⊂

?

same concept as

(pharo list discussion)

- Symbol → uniqueness
  - why don't we name it "UniqueString" for example?
- But **SOME** symbols are "selectors"
  - They answer #isUnary, #isBinary, #precedence, etc
- One class is used to represent two different concepts
  - Knowledge representation problem…

10 Pines

Another Example

If we remove #isBinary, #precedence to symbols, are they still symbols?

"A designer knows he has achieved perfection not when there is nothing left to add, but when **there is nothing left to take away**"

(Saint Exupery)

I think two classes would be better, UniqueString and MessageName (not even Selector… remember the axioms)

So…

- I don't want the programmer to think all the time…
  - "here, is it only a symbol or a message name?"
  - Is this just a string or an identifier (a name)?
  - Is this just a number or an amount of money?
- I want…

computers do more… and people less…

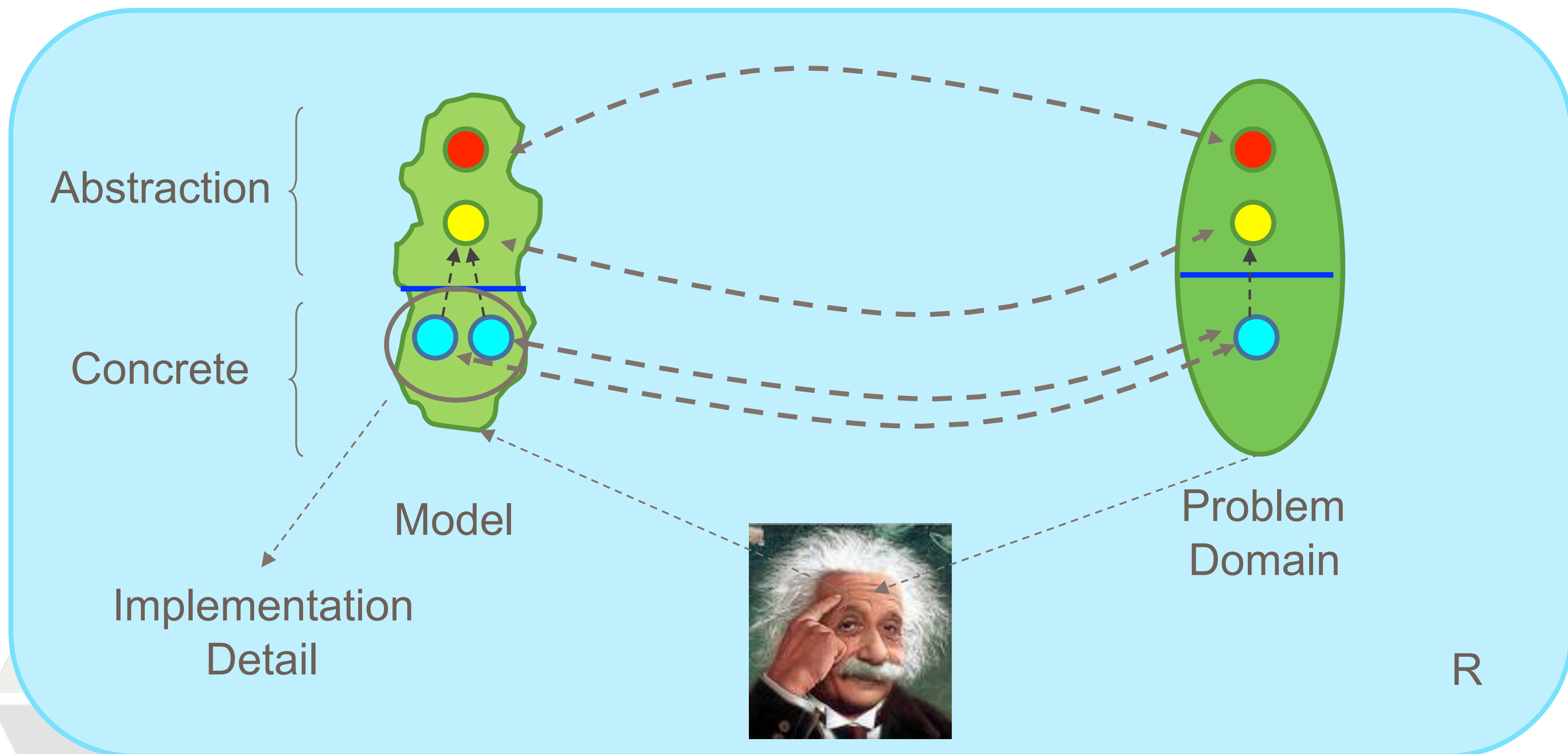
10 Pines

Tip 1:
Try to have an Isomorphism between classes and concepts of the problem domain

10 Pines

Tip 1: Isomorphism

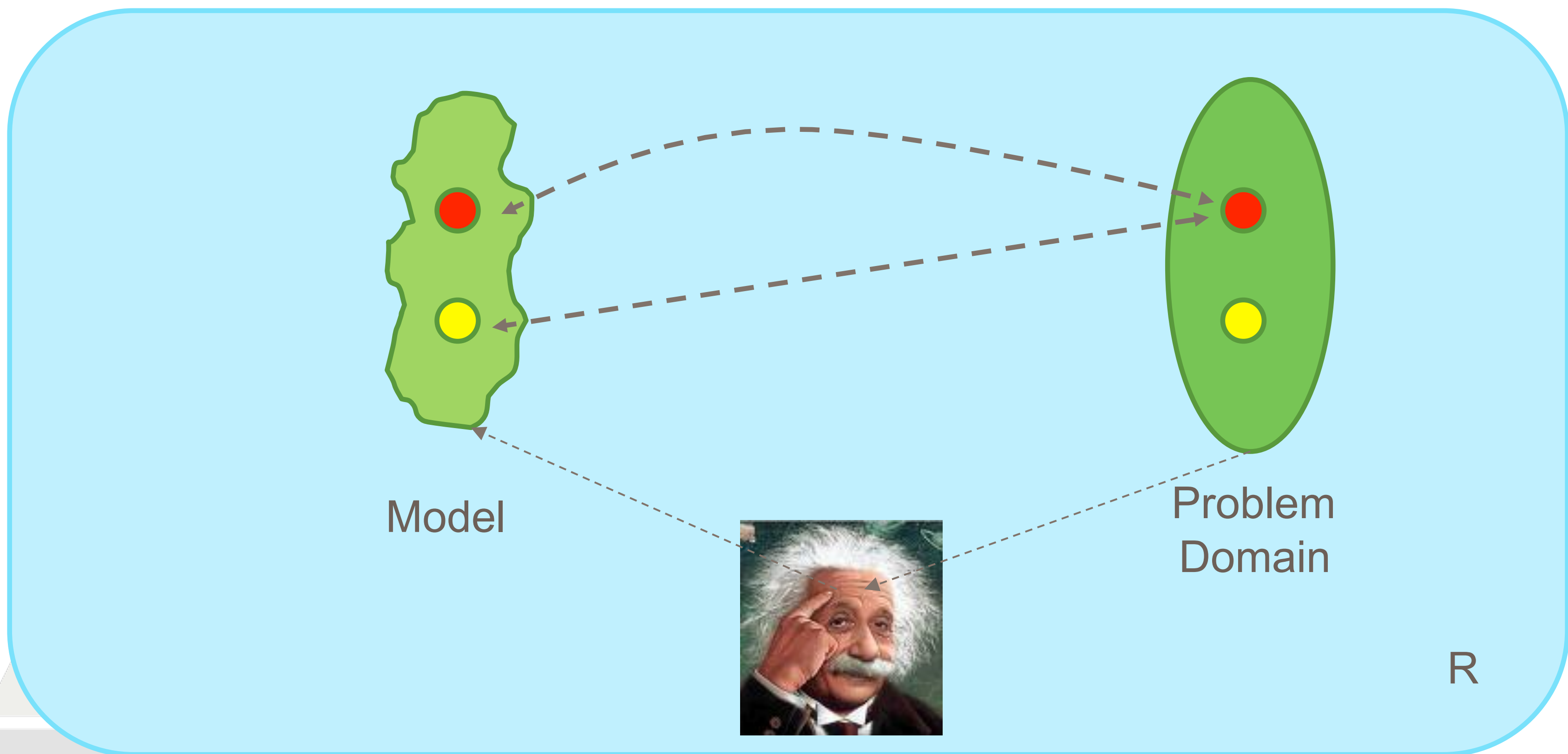
- One to one mapping between classes and concepts



10 Pines

Tip 1: Isomorphism

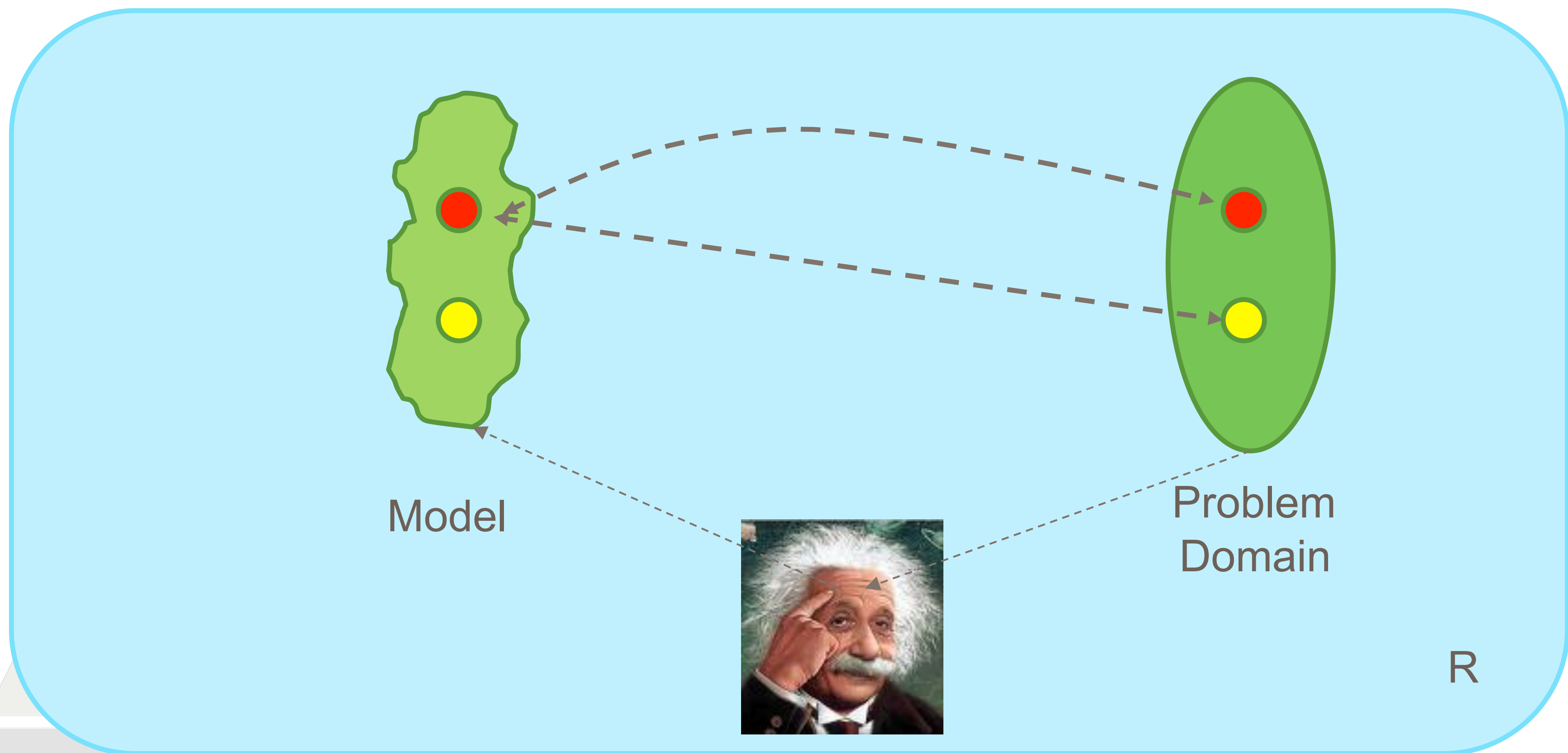
- No two classes for one concept (not common)



10 Pines

Tip 1: Isomorphism

- No one class for two concepts



Examples of one class, many concepts

- Symbol used to represent a day
  - #Monday represents the day Monday, so you can as a day (i.e. Monday) if it is binary ☺
- Number to represent a year
  - Is the number 2008 a good representation of the year 2008?
  - What does it mean the factorial of year 2008?
  - How do you know if year 2008 is leap or not?
- Number to represent a measure
  - Is 10 a good representation for 10 dollars?
- String used to represent names (identifiers)
  - “Hernan” will not be the same as “HERNAN” or “ HeRnAn “
- Just code…
  - Is the code “initialCapital * interestRate * time” a good representation of the Interest received in an investment?
  - Why not having an Interest class?

10 Pines

Tip 1 - Conclusion

- I’m not saying “create a class per each role they play in a context”. Example
  - Numerator is not another class, we use integers to represent a fraction’s numerator, and that integer will play the role of the fraction’s numerator
- Some people say “It is more complex, you will have more classes”
  - Remember Alan Kay?: **“everything is about the verbs”**
  - The number of messages will be the same!
    - #isBinary, #isKeyword, #precedence etc. will be in MessageName but we will not add new messages to it that were not in Symbol
- What we recognize as concept of the problem domain depends on our analysis capabilities and design experience

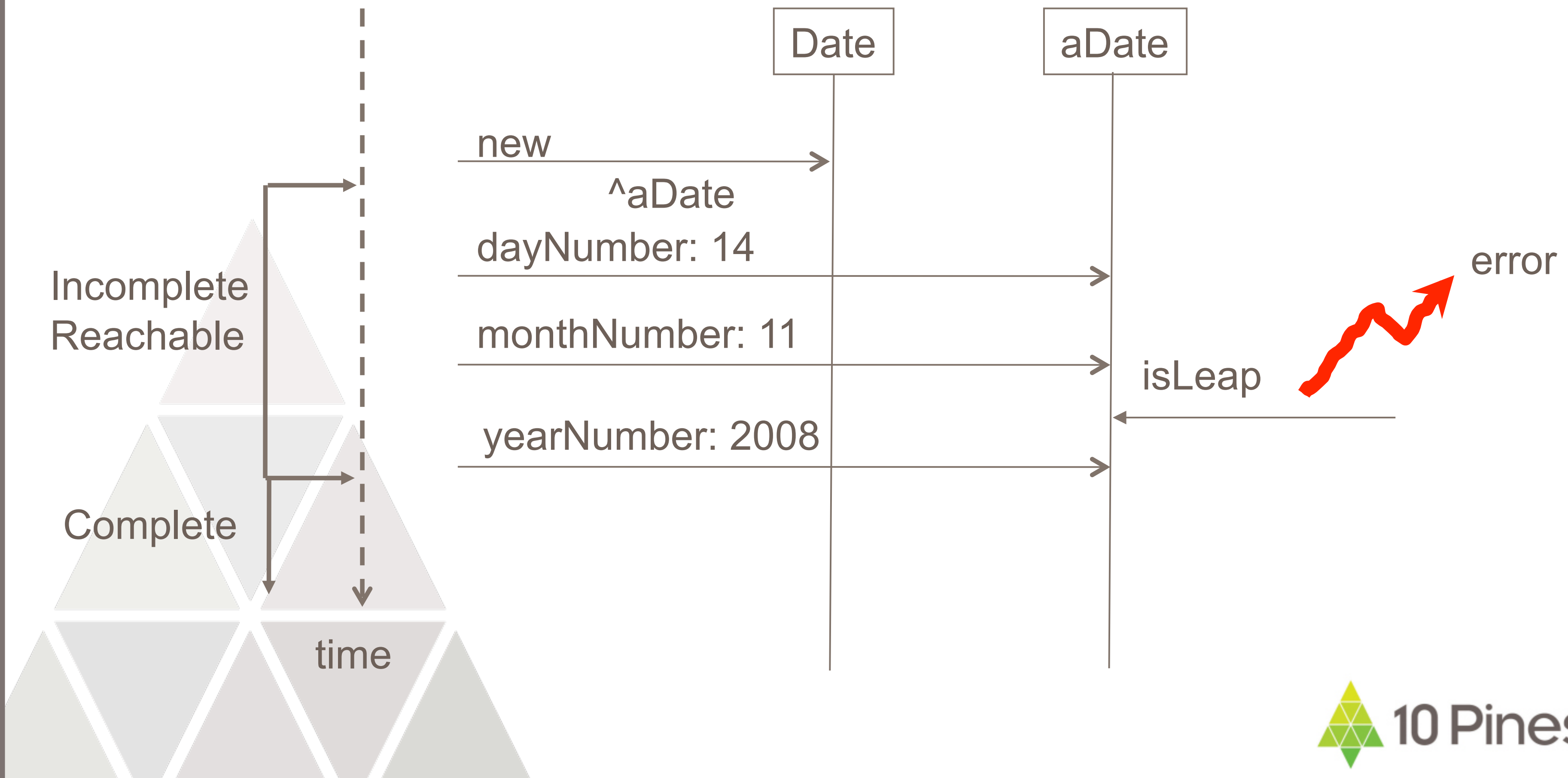
10 Pines

Tip 1 - Conclusion

- Benefits:
  - Cohesion
  - Direct mapping between the model and the problem domain
  - Well defined abstractions
  - Objects with clear responsibilities
  - Easier to understand
- **QUESTION:** How do we create objects?

Classic way to create objects

- Setters



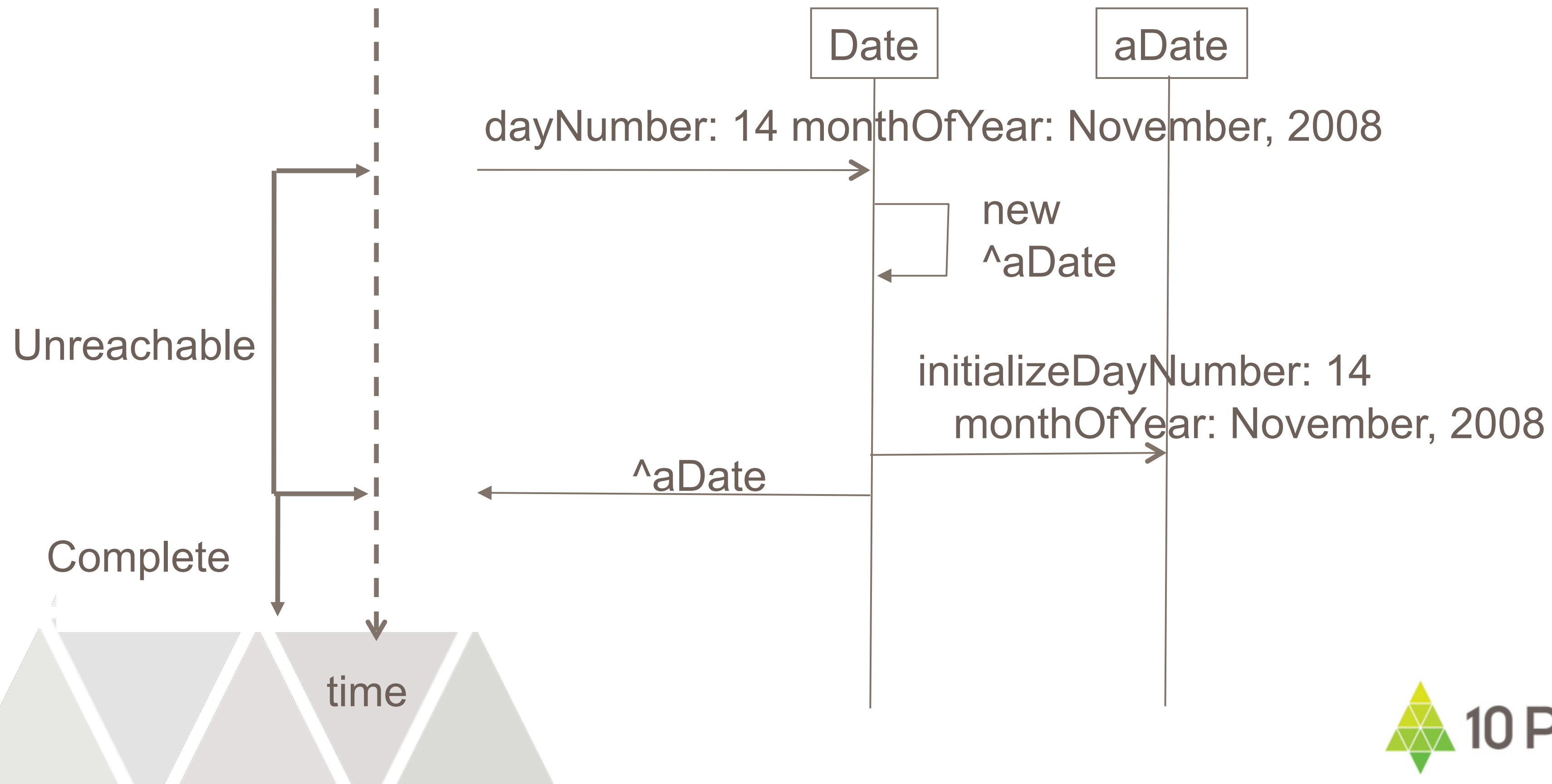
Classic way to create objects

- Disadvantages:
  - The user of the object has the responsibility of "completing" it
  - That responsibility is distributed, error prone, not easy to maintain
  - No easy way to ensure its completeness because nothing define when everything is set
  - There is a time span where the object does not represent anything!
  - Nothing prevents us from changing the object

10 Pines

Proposed way to create objects

- The object must represent the entity since time zero



10 Pines

Tip 2: Objects must be complete since their creation time

10 Pines

Tip 2: Implementation - PatAttendee

browse | hierarchy

```
relatedTo: anUser
  contactedWith: anAttendeeContactInformation
  workingAt: anAttendeeWorkInformati
  chossing: anAttendeeConferenceOptions
  payingWith: anAttendeePayment
  and: anAttendeeAdditionalInformation
  on: aDate

  ^ self new
    initializeRelatedTo: anUser
    contactedWith: anAttendeeContactInformation
    workingAt: anAttendeeWorkInformati
    chossing: anAttendeeConferenceOptions
    payingWith: anAttendeePayment
    and: anAttendeeAdditionalInformation
    on: aDate
```

One instance creation message that will receive all the objects

- One initialization message
- initializeXxx pattern

10 Pines

Tip 2: Implementation - PatConference

```
configuredBy: aConferenceConfiguration

  ^ self
    configuredBy: aConferenceConfiguration
    managingTimeWith: PatRealTimeTimeSystem usingGregorianCalendar
```

All instance creation messages send the "real one"

```
configuredBy: aConferenceConfiguration managingTimeWith: aTimeSystem

  ^ self new
    initializeConfiguredBy: aConferenceConfiguration
    managingTimeWith: aTimeSystem
```

One "real" instance creation message

10 Pines

Tip 2: Advantages

- Objects will always answer no matter "when" the message is sent (they are "complete")
- Programmer will not worry about "can I send this message here/now?" (we are simplifying the protocol!)
- Easy and consistent implementation
  - Only one method does new and send initialize…
  - Pattern: named: xxx → initializeNamed: xxx
  - Easy to check that nobody but the class sends the initialization message

10 Pines

Tip 2: To think about…

- An object that is “filled out” later, should be represented at least by two objects
  - We will see an example…

- **QUESTION:** But, are the objects really “valid”??

10 Pines

Valid Objects

- Should the model check that objects are valid?
- If an object represents an entity of the problem domain, what does an "invalid" object represent?
- If we allow invalid objects to exist, how are the programmers going to learn "from the model"?

10 Pines

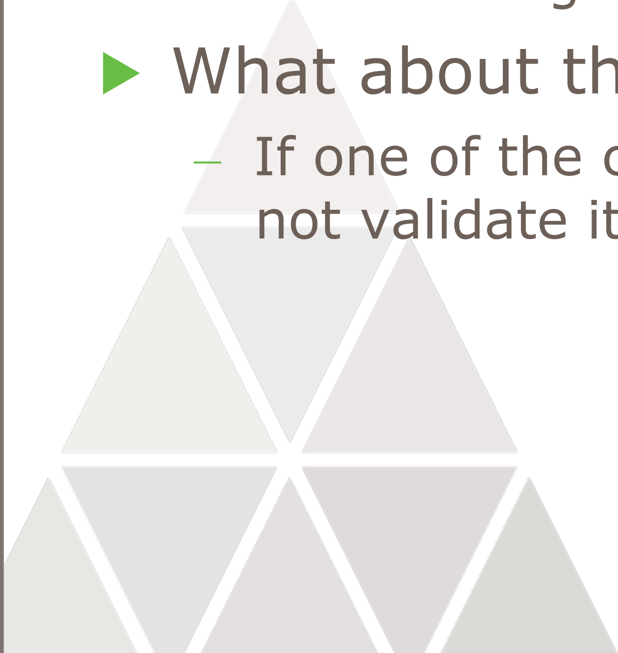
We need “Self Defensive” Objects

- **YES**, our model should be:
  - “self defensive” and
  - “auto documented” about validity rules/ constrains

10 Pines

Where should objects be self defensive?

- On the method that creates the object?
  - Repeated code → Error prone
- Is the UI responsibility
  - that a number has to be > 0?
  - that a string can not be empty?
- What about the instance creation message itself?
  - If one of the class responsibility is to create objects, why not validate its creation first?



10 Pines

Example - PatFee

```
forAllDaysIs: anAllDaysFee forOneDayIs: aOneDayFee

  PatAssertionsRunner
    valueWith: (self allDaysFeeIsPositiveAssertionFor: anAllDaysFee)
    with: (self oneDayFeeIsPositiveAssertionFor: aOneDayFee )
    with: (self allDaysFee: anAllDaysFee isGreatherThanOneDayFeeAssertionFor: aOneDayFee).

  ^ self new initializeForAllDaysIs: anAllDaysFee forOneDayIs: aOneDayFee
```

```
allDaysFeeIsPositiveAssertionFor: aFee

  ^ PatPositiveAssertion
    for: aFee
    failureDescribedBy: 'All days fee should be greater or equal to 0'
```

```
allDaysFee: anAllDaysFee isGreatherThanOneDayFeeAssertionFor: aOneDayFee

  ^ PatLessThanAssertion
    for: aOneDayFee
    and: anAllDaysFee
    failureDescribedBy: 'The all days fee (<2p>) has to be greather than the one day fee (<1p>)'
```

(this is just an implementation example… you could use other implementation…)

10 Pines

Tip 3: Only Allow Valid Objects to Exist

10 Pines

Tip 3: Advantages

- Only **<u>VALID</u>** objects!!
- The model tells us when we make mistakes!! → learning feedback!!
- Easy to use and implement
- **<u>Some</u>** business rules are reified (we will need more)
- We can meta-program on them…
  - Do you want to document when a fee is valid, just look at its creation assertions

10 Pines

Tip 3: Consequences

- If we have only complete and valid objects…
  - Do we need to use nil?
  - All variables reference to objects that are not nil…

10 Pines

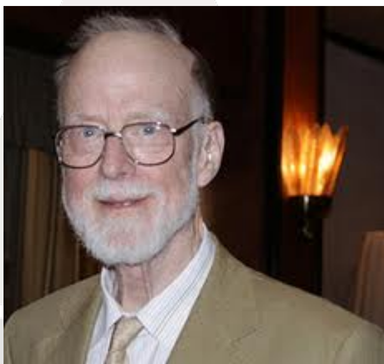
Tip 3: Consequences

No need for nil!!

… a world without nil

It's easy if you try…

you may say I'm a dreamer…



… but I'm not the only one

"Null References: The billon dollar mistake" (Tony Hoare)

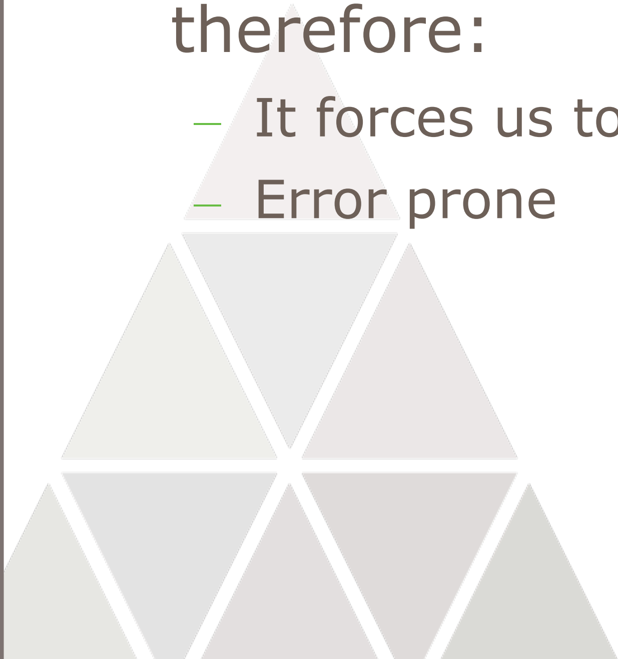
http://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare

Tip 4: Do not use nil

10 Pines

Problems with nil

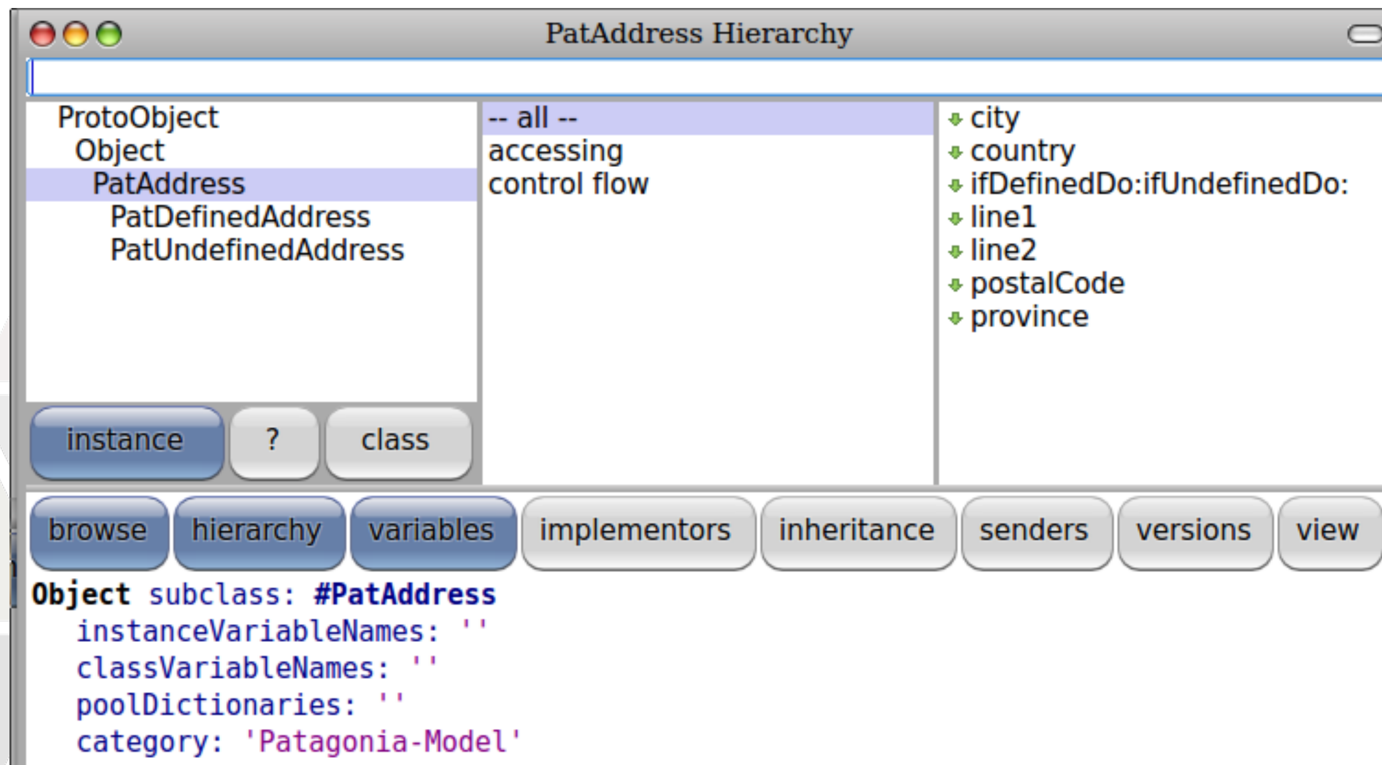
- nil is handy, but has so many meanings that makes it impossible to handle it correctly
  - "Is this variable nil because it was not initialized or was it initialized with nil?"
- nil is not polymorphic with any other object, therefore:
  - It forces us to use "isNil ifTrue:"
  - Error prone



10 Pines

Tip 4 - Advice

- Reify what the absence of something represents
  (null object pattern kind of)
- Never use the "abstract" word in the class name (Thanks Leandro Caniglia!)



10 Pines

Tip 4 – To think about

- Can we have an Image without nil?
- Object superclass → nil
  - Why not NoSuperclass ? (no need with ProtoObject)
- What about NotInitializedVariable instead of nil?

- Remember MouseOverHandler? (lots of problems??)

```
leftMorphs := enteredMorphs := overMorphs := nil
```

```
initializeTrackedMorphs

   leftMorphs := OrderedCollection new.
   overMorphs := WriteStream on: #().
   enteredMorphs := WriteStream on: #().
```

10 Pines

Going back to Tip 3: Consequences

- Can we show assertions description to the user?
  - YES!! → Error descriptions are in one place, with the assertions
  - We should show more than one error description at the same time
  - We should show error descriptions in the right place

to

and

10 Pines

Tip 3: Consequences in the UI

Early Registration Fee

All Days Fee

10

One Day Fee

-1

One day fee should be greater or equal to 0

Late Registration Fee

All Days Fee

11

One Day Fee

12

The all days fee (11) has to be greather than the one day fee (12)

Early Registration Fee

All Days Fee

20

One Day Fee

19

Late Registration Fee

All Days Fee

13

The all day early registration fee (20) has to be less than the all date late registration fee (13)

One Day Fee

12

The early one day registration fee (19) should be less than the late one day registration fee (12)

UI Implementation - PatFeeRendererBuilder

```
createModelHolder

  | allDaysFeeModelHolder oneDayFeeModelHolder |

  allDaysFeeModelHolder := PatWebNumberModelHolder withNumberOfDecimals: 2.
  oneDayFeeModelHolder := PatWebNumberModelHolder withNumberOfDecimals: 2.

  ^ (PatWebGenericCompositeModelHolderBuilder
     for: PatFee
     accessors: #(#allDaysFee #oneDayFee)
     handling: (Array
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #allDaysFee:isGreatherThanOneDayFeeAssertionFor:
           is: oneDayFeeModelHolder)
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #allDaysFeeIsPositiveAssertionFor:
           is: allDaysFeeModelHolder)
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #oneDayFeeIsPositiveAssertionFor:
           is: oneDayFeeModelHolder)))
     forAllDaysIs: allDaysFeeModelHolder
     forOneDayIs: oneDayFeeModelHolder
```

Model holder for each “part”

10 Pines

UI Implementation - PatFeeRendererBuilder

```
createModelHolder

  | allDaysFeeModelHolder oneDayFeeModelHolder |

  allDaysFeeModelHolder := PatWebNumberModelHolder withNumberOfDecimals: 2.
  oneDayFeeModelHolder := PatWebNumberModelHolder withNumberOfDecimals: 2.

  ^ (PatWebGenericCompositeModelHolderBuilder
     for: PatFee
     accessors: #(#allDaysFee #oneDayFee)
     handling: (Array
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #allDaysFee:isGreatherThanOneDayFeeAssertionFor:
           is: oneDayFeeModelHolder)
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #allDaysFeeIsPositiveAssertionFor:
           is: allDaysFeeModelHolder)
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #oneDayFeeIsPositiveAssertionFor:
           is: oneDayFeeModelHolder)))
     forAllDaysIs: allDaysFeeModelHolder
     forOneDayIs: oneDayFeeModelHolder
```

A composite model holder for PatFee

Using same instance creation message

10 Pines

UI Implementation - PatFeeRendererBuilder

```
createModelHolder

  | allDaysFeeModelHolder oneDayFeeModelHolder |

  allDaysFeeModelHolder := PatWebNumberModelHolder withNumberOfDecimals: 2.
  oneDayFeeModelHolder := PatWebNumberModelHolder withNumberOfDecimals: 2.

  ^ (PatWebGenericCompositeModelHolderBuilder
     for: PatFee
     accessors: #(#allDaysFee #oneDayFee)
     handling: (Array
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #allDaysFee:isGreatherThanOneDayFeeAssertionFor:
           is: oneDayFeeModelHolder)
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #allDaysFeeIsPositiveAssertionFor:
           is: allDaysFeeModelHolder)
        with: (PatAssertionFailureToErrorReporterDispatcher
           for: #oneDayFeeIsPositiveAssertionFor:
           is: oneDayFeeModelHolder)))
     forAllDaysIs: allDaysFeeModelHolder
     forOneDayIs: oneDayFeeModelHolder
```

Show errors on the right model holders

The model knows nothing about the UI

10 Pines

UI Implementation - PatFeeRendererBuilder

Difficult to write?? Just write:

```
PatWebGenericCompositeModelHolderBuilder
   for: PatFee
```

Let Smalltalk do the rest…

(Print it…)

```
PatWebGenericCompositeModelHolderBuilder
   for: PatFee
   named: 'PatFee'
   accessors: #( allDaysFee oneDayFee )
   handling: (Array
      with: ( PatAssertionFailureToErrorReporterDispatcher for: #allDaysFeeIsPositiveAssertionFor: is: )
      with: ( PatAssertionFailureToErrorReporterDispatcher for: #allDaysFee:isGreatherThanOneDayFeeAssertionFor: is: )
      with: ( PatAssertionFailureToErrorReporterDispatcher for: #oneDayFeeIsPositiveAssertionFor: is: )))
      forAllDaysIs:  forOneDayIs:
```

10 Pines

How do we test it? - PatFeeTest

```
testFullFeeShouldBeGreatherThanOneDayFee

	self
		shouldnt: [ PatFee forAllDaysIs: 10 forOneDayIs: 9 ]
		raise: #allDaysFee:isGreatherThanOneDayFeeAssertionFor: asExceptionToHandle.

	self
		should: [ PatFee forAllDaysIs: 10 forOneDayIs: 10 ]
		raise: #allDaysFee:isGreatherThanOneDayFeeAssertionFor: asExceptionToHandle.

	self
		should: [ PatFee forAllDaysIs: 10 forOneDayIs: 11 ]
		raise: #allDaysFee:isGreatherThanOneDayFeeAssertionFor: asExceptionToHandle.
```

Easy way to identify errors

No Exception hierarchy explosion

(Look at PatAssertionModel)

10 Pines

Tip 3: More Consequences

No more setters!!

WHAT are you talking about Wilkins!!!

10 Pines

No setters?

- We cannot have setters because
  - An object could become invalid after a set
  - Need to check validity on setters
    - duplicated code
    - Sometimes it is impossible (i.e. Date)
- But things do change!
  - Do they really?...
  - How often?
  - The truth is most of the things are immutable (or we can model them like that)

10 Pines

Tip 5: Think about immutable objects

10 Pines

Immutability

- When does an object have to be immutable?
  - When the entity it represents is immutable!!
- Examples:
  - Number, Date, Time, Measure, etc
  - Invoice (in Argentina it can not change once the system generates it)
- Advantages
  - No referential transparency problem

10 Pines

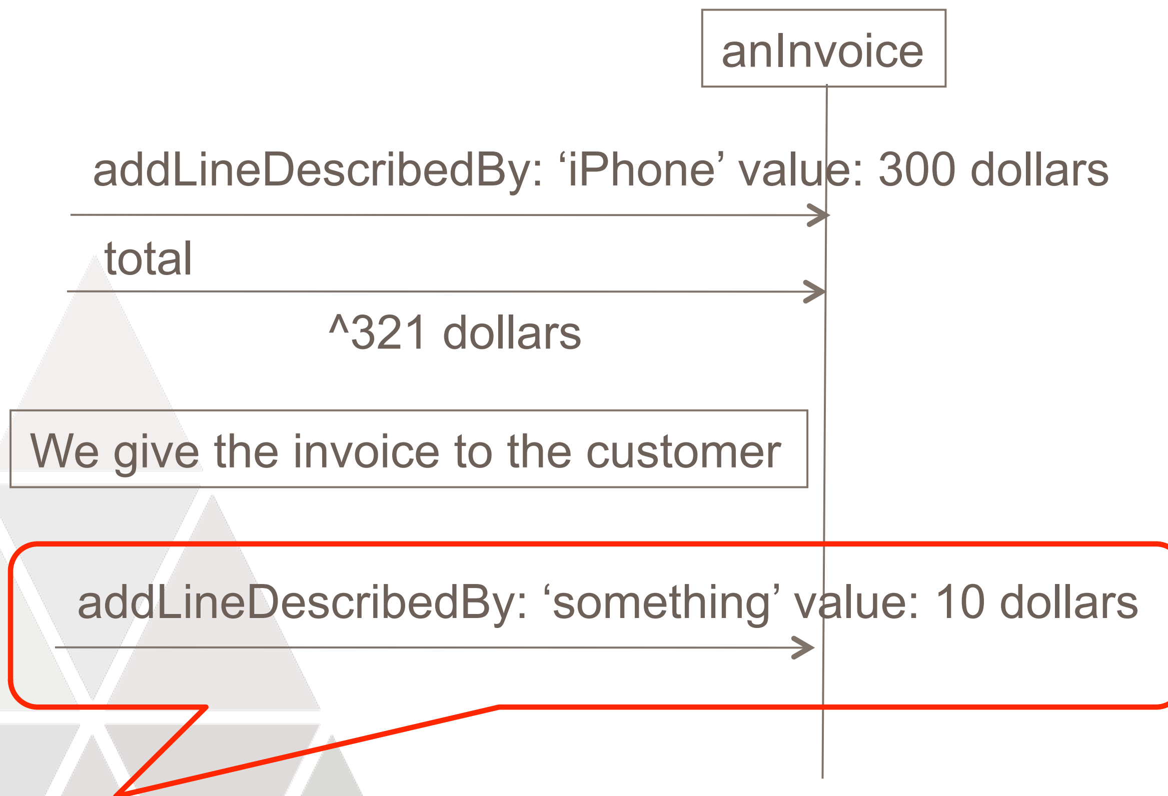
But some things do change…

- Things that are supposed to be fixed but change:
  - An Attendee can change his registration
  - A talk can change its name
  - A Reduction ticket can change its discount %
- How do we model “the change”?
  - Within the same object (classic solution) ?
  - With a relation {change, point in time} ?
  - Create a new object as a result of an event ?
- Remember: Identity does not change!

10 Pines

Case 1: Create a new object as a result of an event

- Bad invoice model
  - Invoice cannot change in Argentina

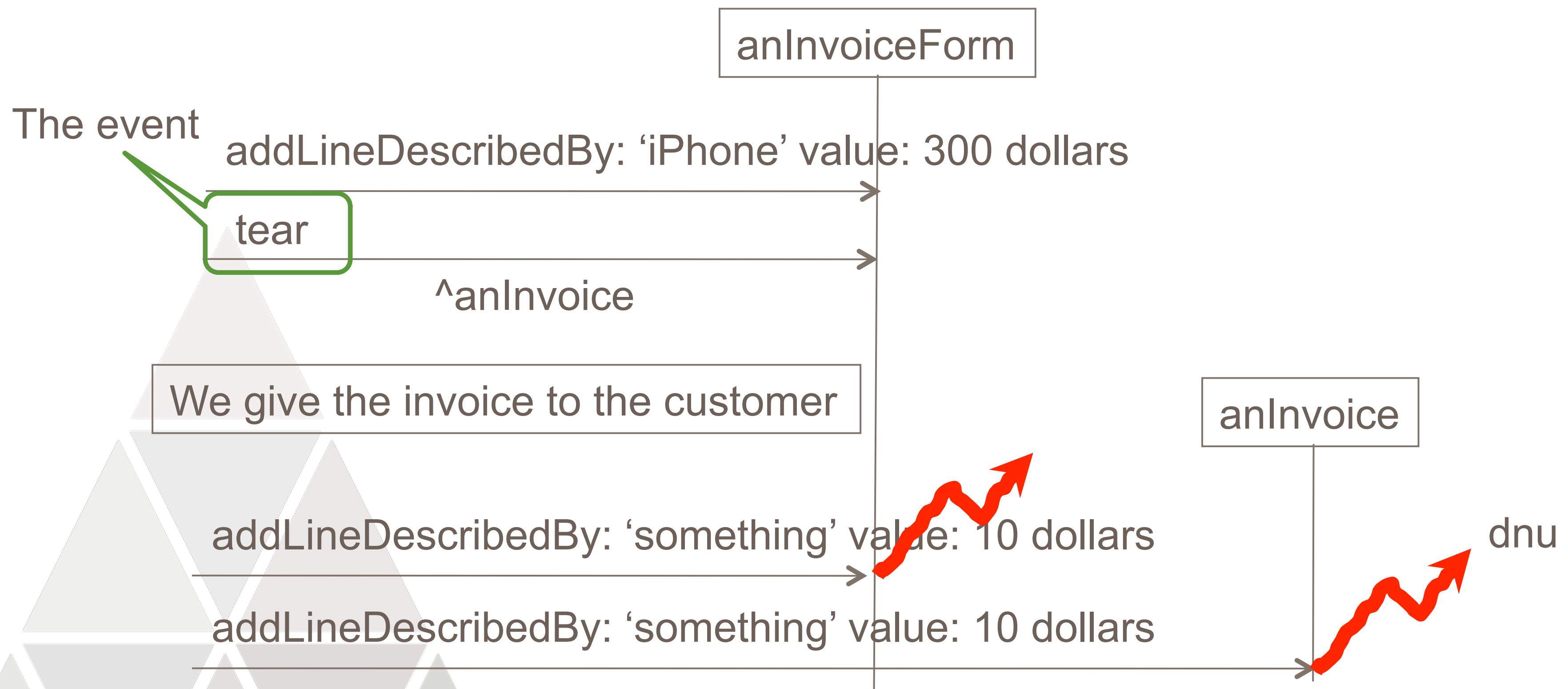


How do we prevent this for happening?!!

10 Pines

Case 1: Create a new object as a result of an event

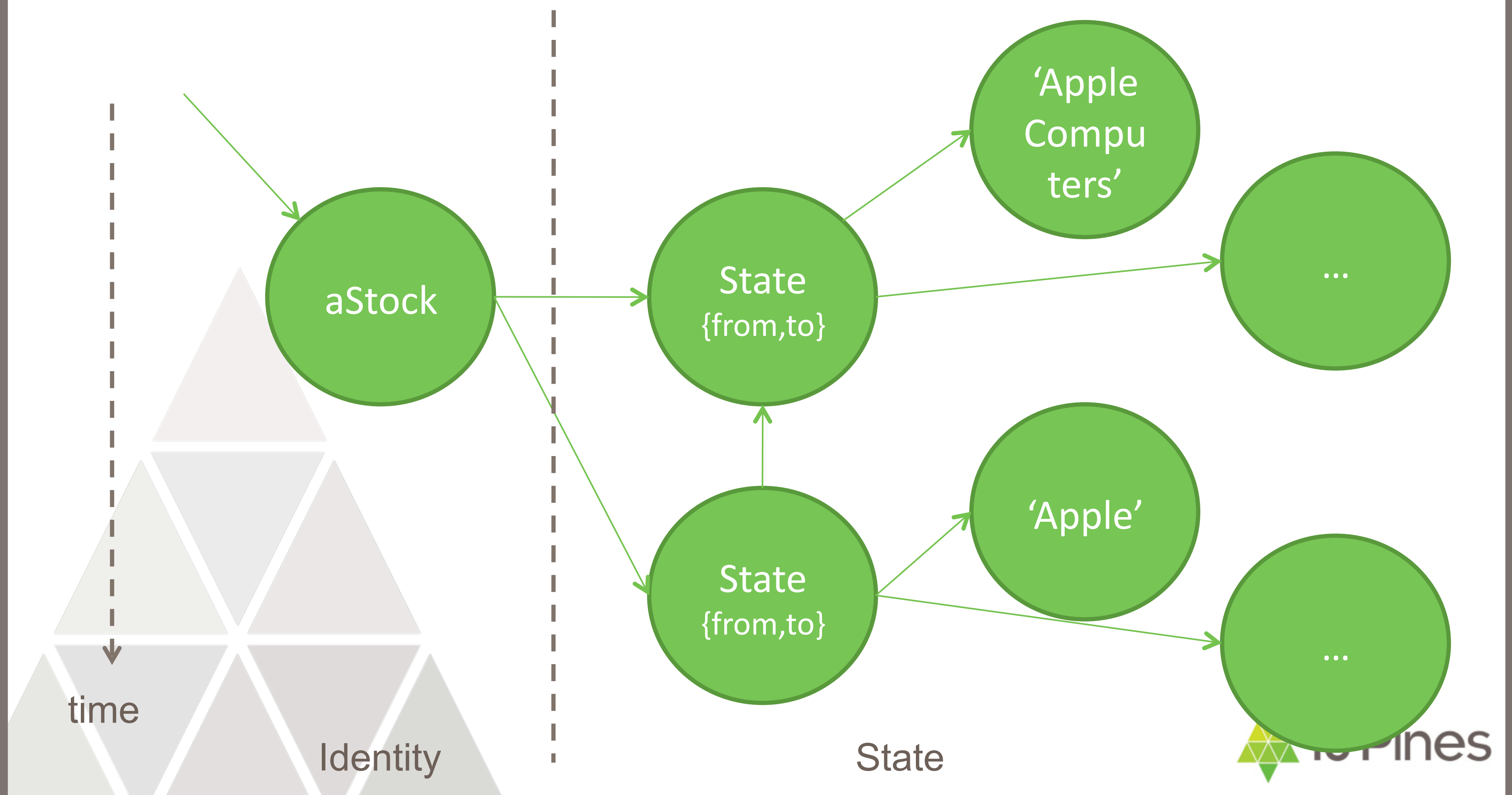
- Good invoice model
- Make time passing explicit



10 Pines

Case 2: Relation {change,time}

- Convert systems from snapshots to “movies”



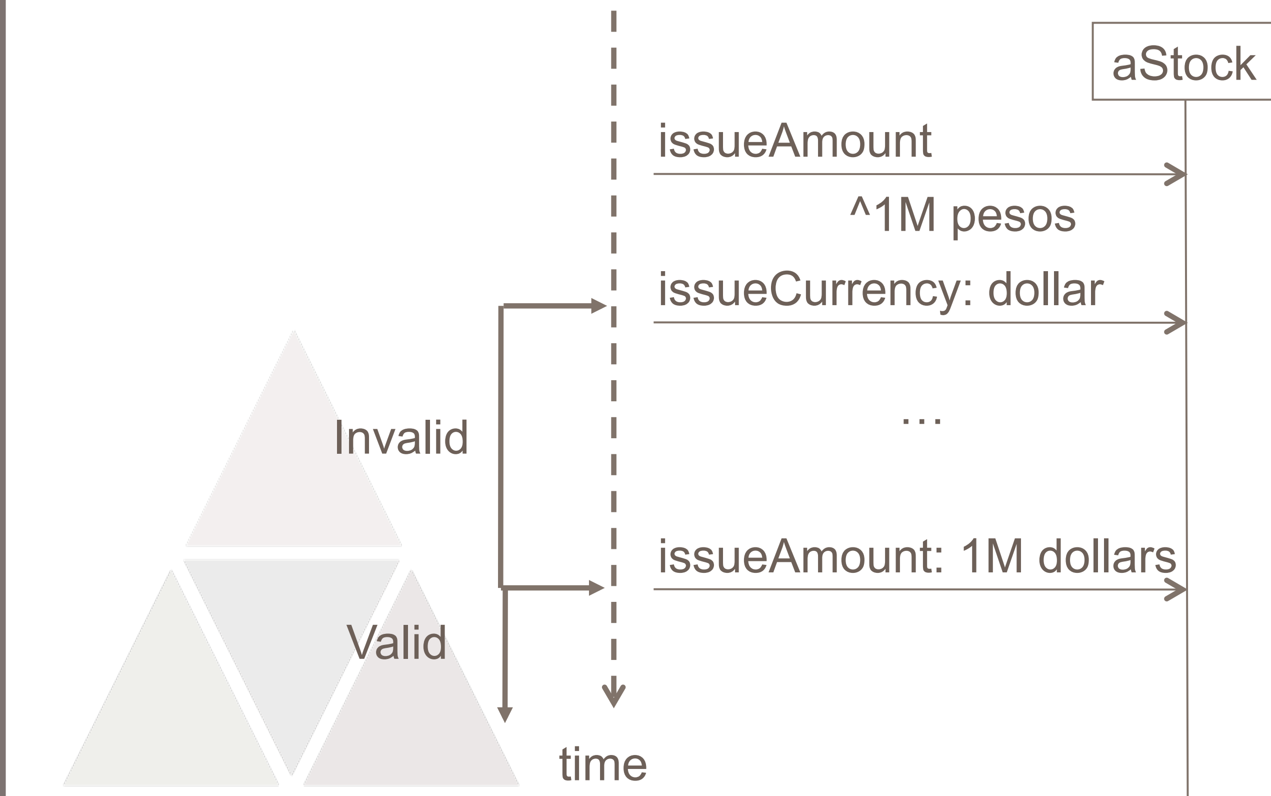
Case 2: Relation {change,time}

- Advantages
  - Time queries
  - Persistence
  - Transaction
  - Audit
  - Makes time pass explicit
- Disadvantages
  - New paradigm
  - New execution model?
  - New meta-model?

10 Pines

Case 3: Using the same Objects

- Classic solution

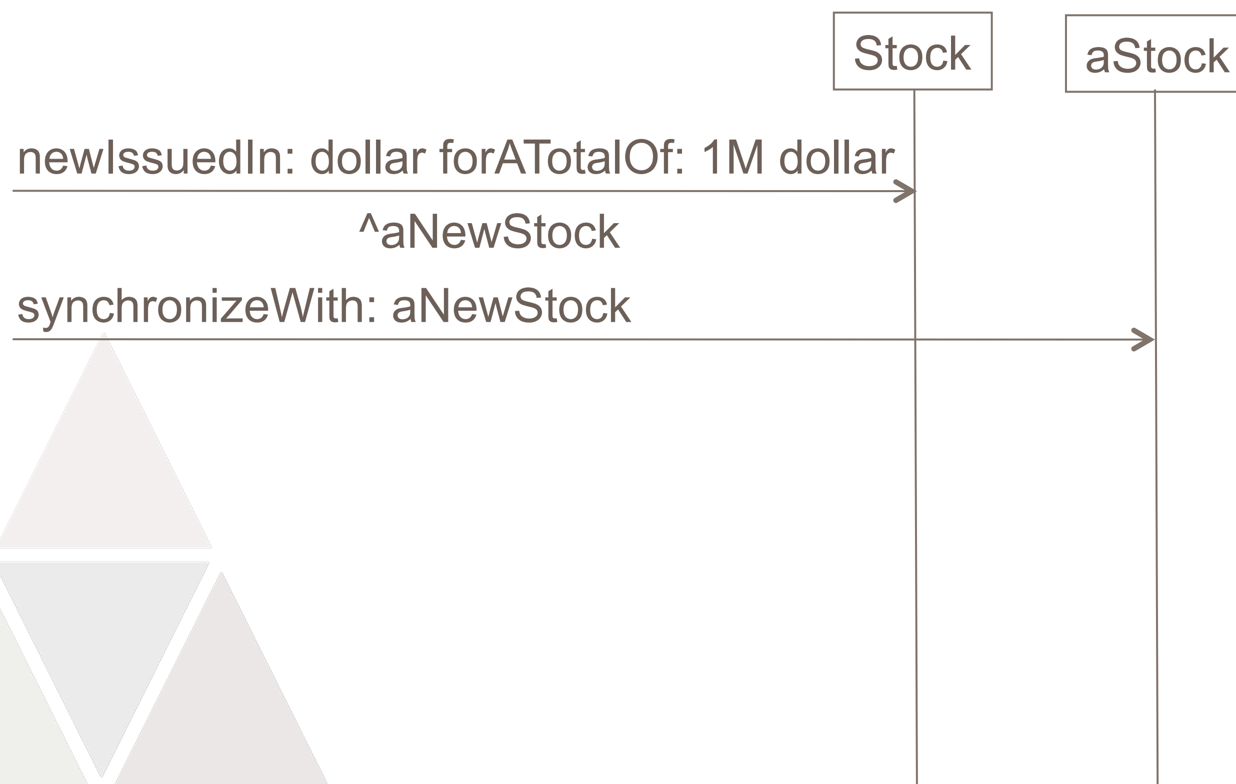


- Same problem as object creation

10 Pines

Case 3: Using the same Objects

- Proposed solution, synchronize at once!



- Same advantages as object creation

10 Pines

Tip 6: Change objects synchronizing at once

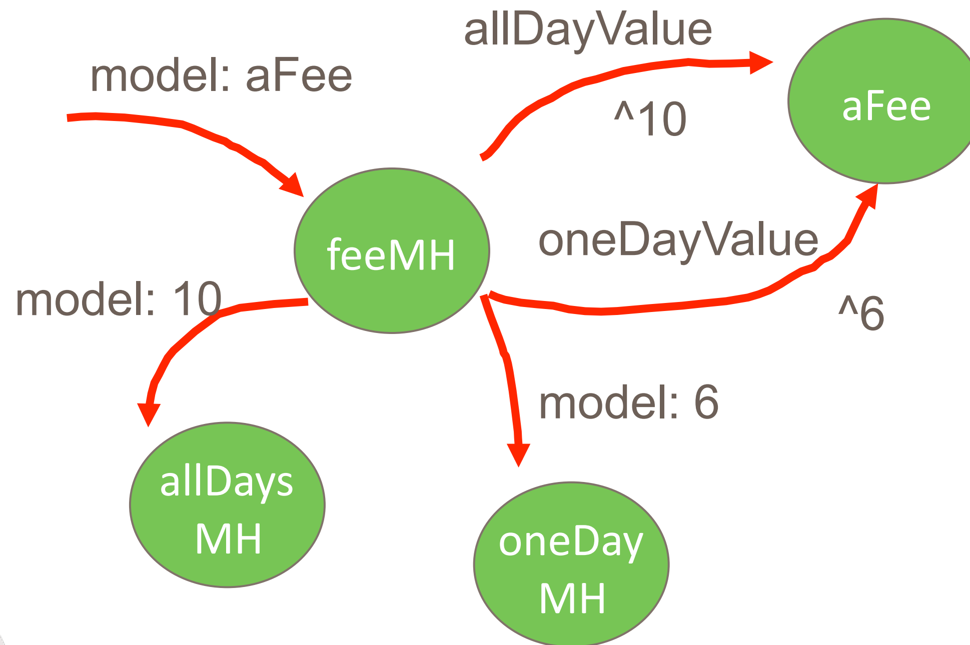
10 Pines

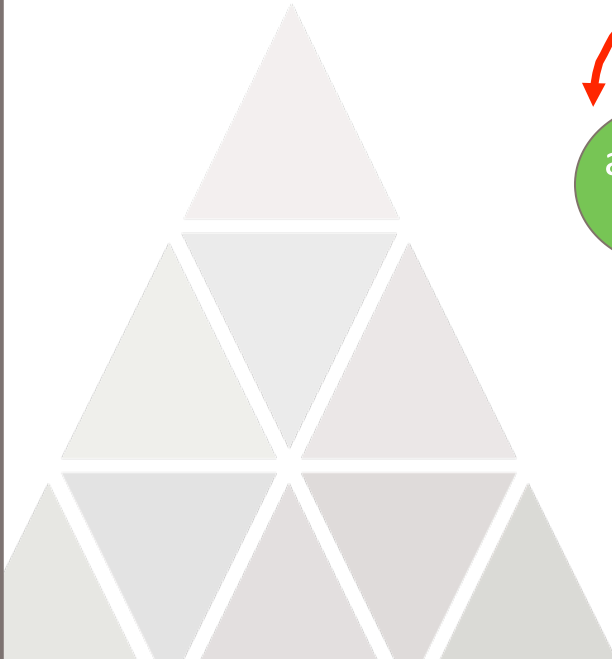
Tip 6: Change objects synchronizing at once

- New objects are created to reflect the changes
- The original object synchronizes with the copy using just one message (synchronization message or object)
- Advantages:
  - Business rules are kept in one place: instance creation message
  - Objects are always valid!
- No setter means objects cannot be used as "data holders"
  - This is common in the UI
  - Solution: "model holders" as we saw before

10 Pines

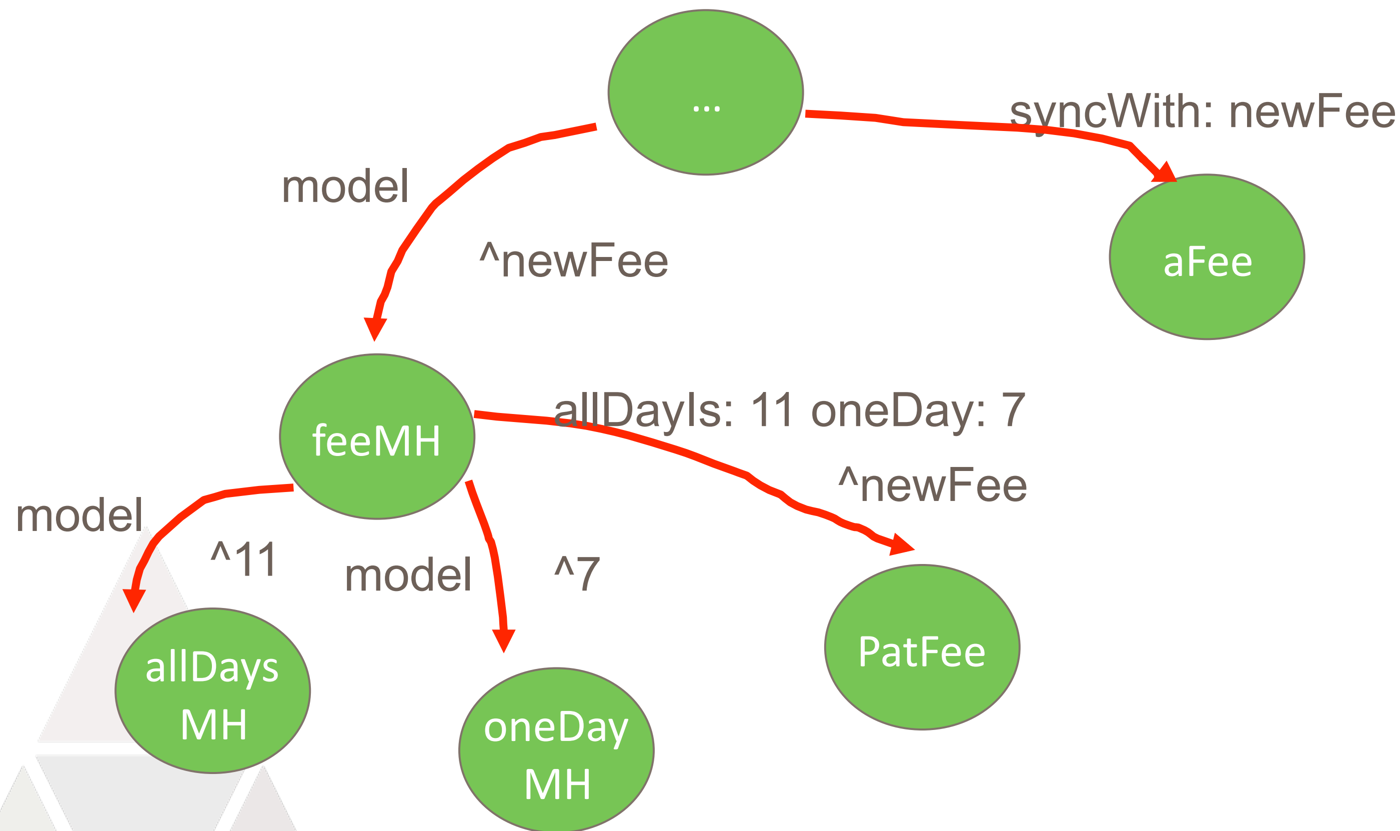
How it works on the UI – Setting the model





10 Pines

How it works on the UI – Getting the model



- No special "object copy", buffering, etc.
- If the user cancels, nothing gets changed
- Validations are done by model's objects

10 Pines

… meaning of an object "in the system"?

- Model objects that handle the sets of cohesive objects
- I call these object SubSystems. For example:
  - PatUserSystem
  - PatAttendeeSystem
  - PatPaymentSystem
  - PatTalkSystem
  - PatAwardSystem
  - and of course… PatPatagoniaSystem
- They have the responsibility of verifying the business rules of an object "in the system"

10 Pines

Tip 7: Model system architecture

10 Pines

Tip 7 – Example of PatAttendeeSystem

```
register: anAttendee

  PatAssertionsRunner
     valueWith: (self validUserAssertionFor: anAttendee user)
     with: (self noAttendeeRegisterWithUserAssertionFor: anAttendee user)
     with: (self attendanceDatesIncludedOnConferenceDatesAssertionFor: anAttendee attendanceDates)
     with: (self attendeeValidCountryAssertionFor: anAttendee country)
     with: (self reductionTicketCanBeUseAssertionFor: anAttendee reductionTicket).

  attendees add: anAttendee.
  self sendWelcomeMailSilentlyTo: anAttendee

  ^ anAttendee
```

Verifies system constrains

Does the real work

Tip 7 – Example of PatAttendeeSystem

```
modifyAttendee: oldAttendee with: newAttendee

  PatAssertionsRunner
    valueWith: (self attendeeIsRegisteredAssertionFor: oldAttendee)
    with: (self attendanceDatesIncludedOnConferenceDatesAssertionFor: newAttendee attendanceDates)
    with: (self attendeeValidCountryAssertionFor: newAttendee country)
    with: (self attendanceDatesOf: newAttendee canNotChangeIfPaidAssertionFor: oldAttendee)
    with: (self reductionTicket: newAttendee reductionTicket canBeUseOrUsedByAssertionFor: oldAttendee).

  ^ oldAttendee syncWith: newAttendee
```

Verifies system constrains

Does the real work

10 Pines

Tip 7 - Advice

- Very important for concurrency
- Check everything… even though you know it cannot happen due to UI implementation
  - Because you know TODAY'S UI implementation!, somebody can change it!
  - The UI is not the only interface of your system! (Rest, WebServices, etc.)

```
submitTalk: aTalk

  PatAssertionsRunner
    valueWith: (self talkNotDuplicatedAssertionFor: aTalk)
    with: (self submissionCreatedByAttendeeAssertionFor: aTalk)
    with: (self talkSubmittedBeforeSubmissionDeadlineAssertion).

  submittedTalks add: aTalk
```

10 Pines

Tip 7 – How does affect tests?

```
testAttendeeCanSubmitATalk

   | esug john |

   esug := testObjectsFactory setUpEsug.
   john := testObjectsFactory john.
   esug register: john.

   self
      shouldnt: [ esug submitTalk: testObjectsFactory johnTalk1 ]
      raise: #submissionCreatedByAttendeeAssertionFor: asExceptionToHandle.
   self assert: (esug numberOfSubmittedTalksBy: john) = 1.
```

Force you to have the right setup

Need for factory objects

```
testTalksShouldBeSubmittedByAttendees

   | esug |

   esug := testObjectsFactory setUpEsug.

   self deny: (esug existAttendeeIdentifiedAs: testObjectsFactory ringo email).
   self assert: esug numberOfSubmittedTalks = 0.
   self
      should: [ esug submitTalk: testObjectsFactory ringoTalk1 ]
      raise: #submissionCreatedByAttendeeAssertionFor: asExceptionToHandle.
   self assert: esug numberOfSubmittedTalks = 0.
```

10 Pines

Tip 7 - Advantages

- Objects are complete and valid (previous tips)
- Objects in the system are valid
- "One system to rule them all"
  - PatPatagoniaSystem
    - It is the one registered in the SeaSide application
    - It is the root for all objects in GemStone
- We can do meta-programming on the architecture!
- We can have different system implementations
  - i.e. AuthenticationSystem
- We can distribute the systems…

10 Pines

Tip 7 – Advice: Model time system

- Tests have to control EVERYTHING, also the time…

PatTimeSystem Hierarchy

Classes	Categories	Methods
ProtoObject	-- all --	currentYear
Object	accessing	initialize
PatTimeSystem	initialization	moveNowForward
PatRealTimeTimeSystem	test support	moveToNextDay
PatTestTimeSystem		now
		now:
		today

instance ? class

browse hierarchy variables implementors inheritance senders versions view

browse senders of…

```
now: aDateTime

   now := aDateTime
```

10 Pines

Tip 7 – Advice: Model time system

```
testTalksCanNotBeSubmittedAfterSubmissionDeadline

   | esug john |

   esug := testObjectsFactory setUpEsug.
   john := testObjectsFactory john.
   esug timeSystem now: esug submissionDeadline next atMidnight.

   self assert: esug numberOfSubmittedTalks = 0.
   self
      should: [ esug submitTalk: testObjectsFactory johnTalk1 ]
      raise: #talkSubmittedBeforeSubmissionDeadlineAssertion asExceptionToHandle.
   self assert: esug numberOfSubmittedTalks = 0.
```

For the tests, PatPatagoniaSystem uses a PatTestTimeSystem

We control the time to test "time" related constrains/events

10 Pines

There are more things to talk about like…

- Assertions model
- Patagonia WebUI Model (not as nice as I would like)
- But for sure we ran out of time…

10 Pines

Conclusions

- Axiom 1: Software as computable MODEL
- Axiom 2: Software development as LEARNING PROCESS
- Tip 1: Isomorphism between classes and concepts
- Tip 2: Objects must be complete since its creation time
- Tip 3: Verify domain rules at creation time
- Tip 4: Do not use nil
- Tip 5: Think on immutable objects
- Tip 6: Synchronize objects at once
- Tip 7: Model system architecture

- Create your culture around these tips
- Meta-Tip: Use tests to verify the use of these tips

10 Pines

Follow these tips and…

And you will Sleep tight

because you will have just 1 error in your system…

10 Pines

Questions?

10 Pines

agile software development & services