

The Impact of XML on Software Testing and Software Quality

*Stuart Baurmann,
CTO, Scrutable Systems*

www.scrutable.com

September, 2001

How is XML changing enterprise software implementation ?

Messaging and Integration

*Content Management and
Presentation*

XML documents

Database Access

XML application data

System Configuration

XML Messaging and Integration

Message transfer interfaces

SOAP, WSDL, UDDI

Message schemas and standards

XML Schema 1.0 – released in May

OASIS/ebXML and Microsoft BizTalk

Both now based on SOAP

Domain specific efforts

openhealth.org, opentravel.org, etc.

Message Mapping and Routing

Conversion of XML to/from EDI, CSV, etc.

Mapping SOAP to CORBA, EJB – XMLBus, WebSphere

XSLT Transformations

Complete XML Solutions: Webmethods, eXcelon, etc.

XML Content Management and Presentation (for documents)

What can be stored, displayed, edited ?

Specification Documents

Stylesheets

Schemas

Test plans and results !!!

Presentation options

HTML, WML, SVG, Flash

Content Management Tools

RDF, WebDAV

Document Persistence:

Solutions (Vignette, Arbortext)

Repositories (Tamino, eXcelon, Virtuoso)

Adapters (Oracle, IBM, Microsoft)

XML Database Access (for application data)

Accessing relational data

Oracle XSQL

Microsoft SQL server updategrams

“Native” XML storage

Tamino, eXcelon, xyzFind, Ipedo, dbXML

Examine update interfaces carefully

Hybrid Solutions

OpenLink Virtuoso

Forthcoming Microsoft offerings

Emerging Standards

XML:DB

XUpdate

XML Configuration files

For off the shelf tools:

ANT build files

EJB deployment descriptors

WAR application deployment files

Creating your own

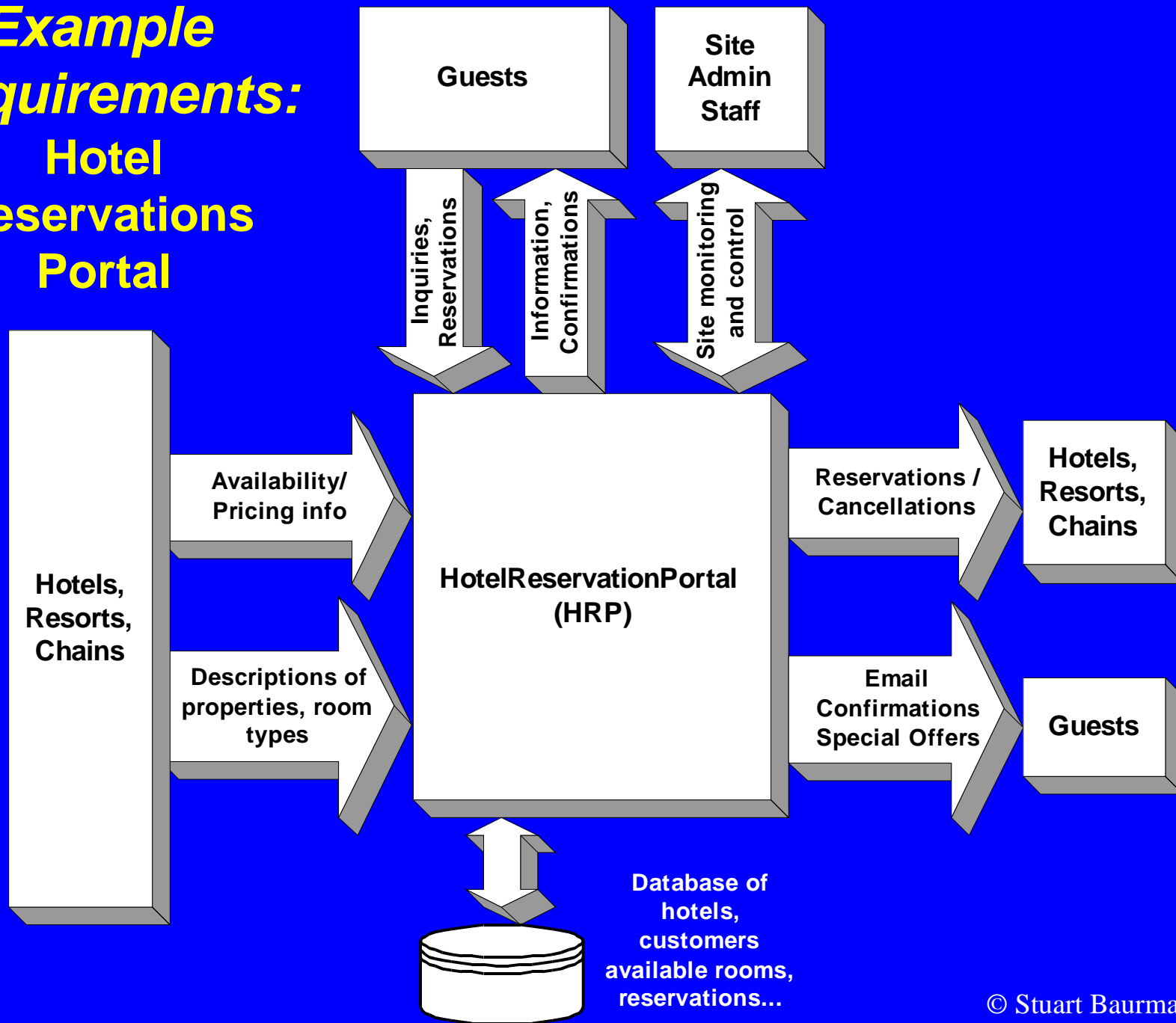
Java Parsers and APIs

Apache Xerces, JDOM, DOM4J

Java Data-binding

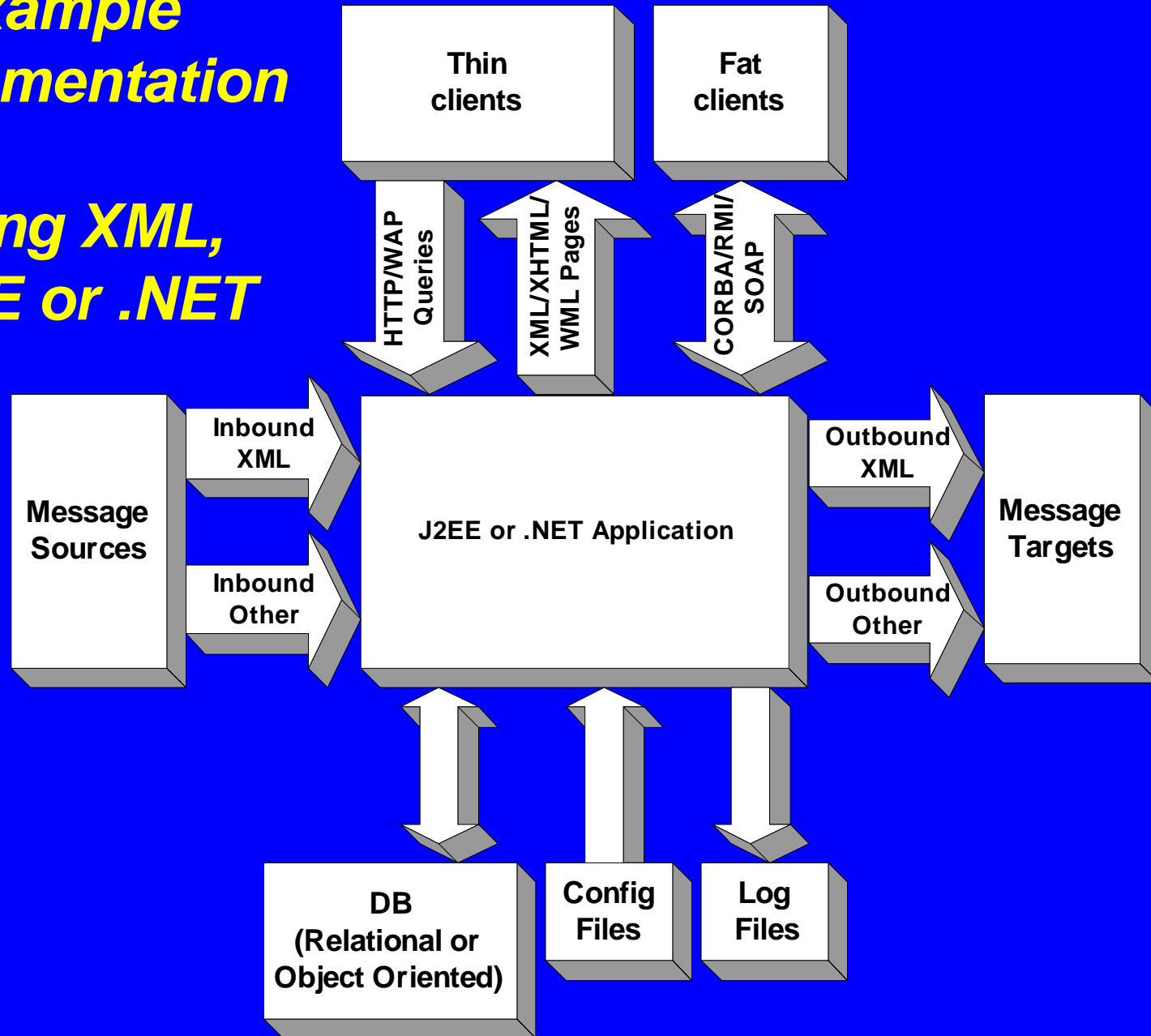
Sun JAXB, JSX, Breeze

Example Requirements: Hotel Reservations Portal



Example Implementation

using XML,
J2EE or .NET



Ex: Describing a Hotel

```
<Hotel ID="US-TX-AUS-RadissonArboreteum">
  <FullName>Radisson Austin Arboreteum</FullName>
  <ChainName>Radisson</ChainName>
  <LocationName>Austin Arboreteum</LocationName>
  <LocationAddress>1000 Capital of Texas Hwy</LocationAddress>
  <LocationCity>Austin, TX</LocationCity>
  <NumberOfRooms>450</NumberOfRooms>
  <Rating>***</Rating>
  <AAA Discount>YES</AAADiscount>
  <RestaurantOnSite>YES</RestaurantOnSite>
  <Pool>YES</Pool>
  <Picture>&referenceToHotelPictureEntity;</Picture>
  <RoomTypeList>
    <RoomType ID="DLX-KING-OCN-NSMK">
      <Description>Deluxe King Bed, Ocean-View, No Smoking</Description>
      <Grade>Deluxe</Grade>
      <Beds>
        <Bed type="King">
        </Beds>
      <View>Ocean</View>
      <Smoking>NO</Smoking>
      <CoffeePot>YES</CoffeePot>
      <Picture>&referenceToDeluxeRoomPictureEntity;</Picture>
    </RoomType>
    <RoomType ID="STD-QEN2-GAR-SMK">
    ...
  </RoomTypeList>
</Hotel>
```

Ex: Notification of room availability

```
<RoomAvailability ID="RAD2349829384">  
  <HotelID>US-TX-AUS-RadissonArboreteum</HotelID>  
  <RoomTypeID>STD-QEN2-GAR-SMK</RoomTypeID>  
  <StartDate>2001-OCT-05</StartDate>  
  <EndDate>2001-NOV-19</EndDate>  
  <MinimumAdvanceDays>7</MinimumAdvanceDays>  
  <WeekdayPrice>69.00</WeekdayPrice>  
  <WeekendPrice>59.00</WeekendPrice>  
</RoomAvailability>
```

Testing Concepts and Terminology

Why do we need the best possible testing ?

The cost of a defect is exponential in time-to-discover

Approaches to testing

Black box

Functional test, based on specifications

Independent of system implementation

White box

Structural test, makes use of system knowledge

Gray box

Data tests vs. state tests

Testing Concepts Cont...

Levels of testing

Unit testing

Integration testing

System testing

Some other kinds of testing

Performance

Volume

Stress

Regression

Usability

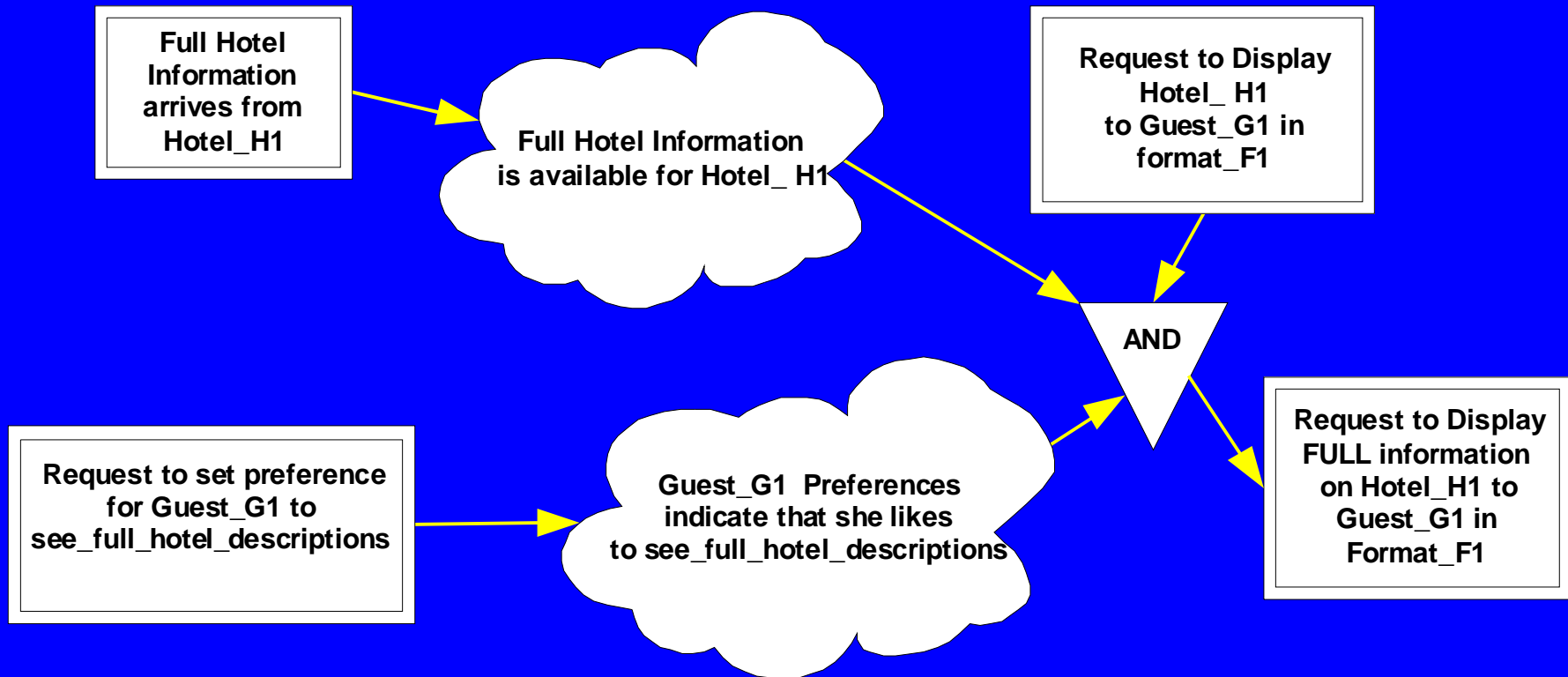
Automated Testing

- **Benefits vs. manual testing**
 - *More bugs are found more quickly*
 - *Regression, Performance, Stress testing is easier*
 - *Tests are more repeatable*
 - *Specifications are forced to be rigorous*
- **Costs**
 - *Acquiring or building tools, training testers*
 - *Integration with other manual/automated processes*
 - *Continuous updates to versioned specifications*
- **Risks (management issues)**
 - *Enslavement under tool's methodology*
 - *Pretty results from poorly designed tests*
 - *Poor supplementary manual coverage*

Automated Black Box Testing

- Pursue a rigorous abstract system specification
 - 1) Define input and output message pathways
 - (Special Case: Request/Reply pairs – e.g Web)
 - 2) Define requirements in terms of constraints on these message pathways
(Cause–effect graphs – “◆EG”◆)
- Define test cases to verify this specification under all foreseeable conditions

Cause-Effect Diagram Example



Hotel	Guest	GuestPreferences	Format
boolean is_full_info_available		boolean see_full_hotel_descriptions	boolean isXML boolean isHTML

Automated Test Cases

What steps does an executable test case need to specify ?

Preconditions/Initialization

Input data generation

Triggering execution

Output data capture

Result/Postcondition validation

Cleanup

Leveraging XML for automated testing

System Description

How do we use XML to define the system interfaces and behaviors so that they are testable ?

Test execution infrastructure

How do we construct test harnesses ?

Test data management

How do we generate, collect, and archive test data ?

Test results validation

How do we determine whether a test succeeded or failed ?

Test planning

Putting the pieces together

XML System Description

Leverage existing resources

Message Schemas / DTDs

Describe data formats that can be used for automatic validation

Schemas are easier to process

WSDL definitions

Describe SOAP, HTTP, MIME message endpoints

XMI models (for non-XML interfaces)

Export from existing UML models.

Describe interfaces: Corba, RMI, EJB, DCOM

Requirements are partially described by use cases

Database metadata

System Configuration Data

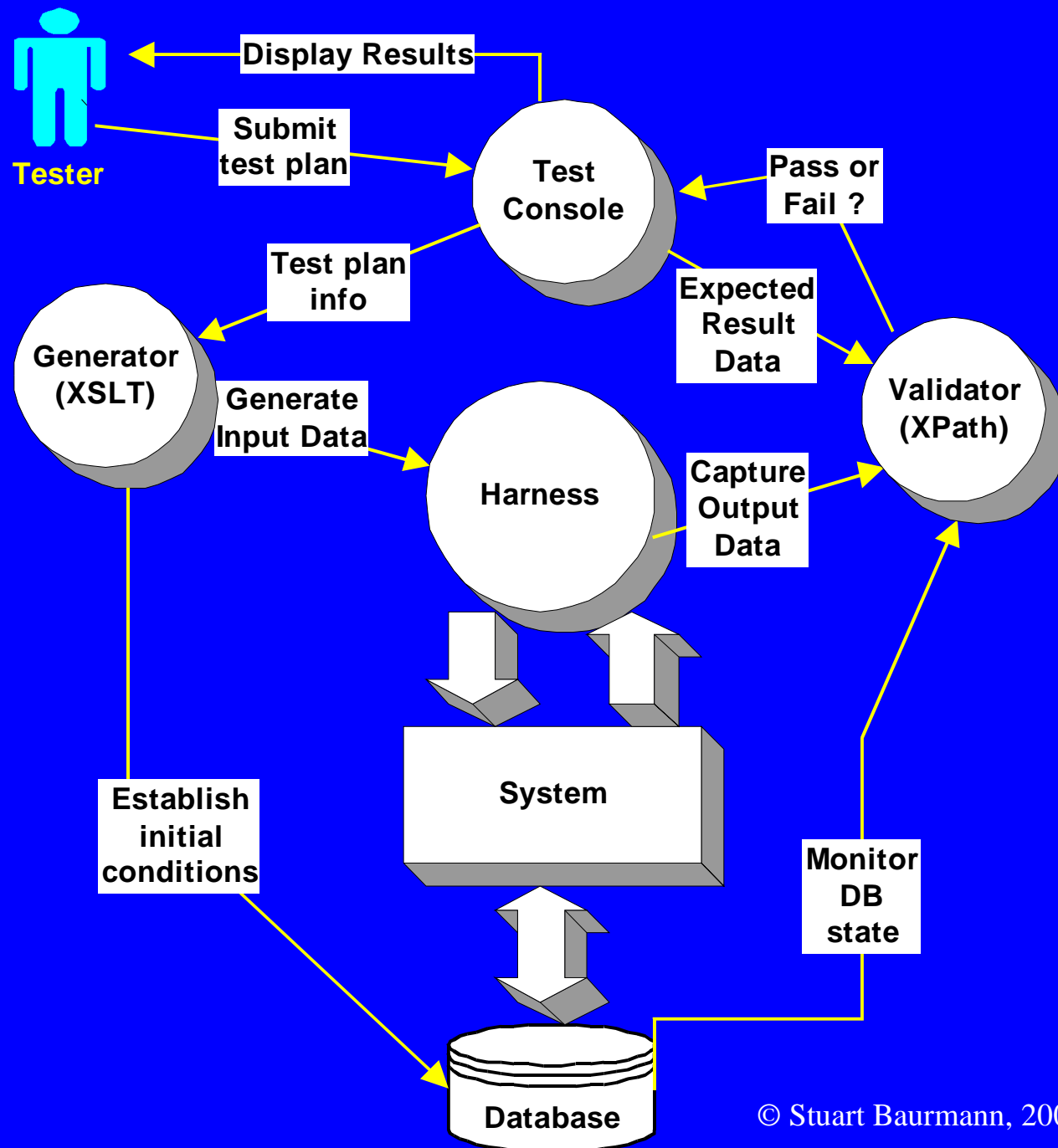
WAR files, EJB deployment descriptors, etc.

Make the import process repeatable !

Example WSDL System Description

```
<message name="UpdateRoomAvailabilityRequest">
  <part name="updatedInfo" type="s0:RoomAvailability" />
</message>
<message name="UpdateRoomAvailabilityResult">
  <part name="errorList" type="s0:UpdateErrorList">
</message>
<portType name="AvailabilityUpdateSoap">
  <operation name="UpdateRoomAvailability">
    <input message="s0:UpdateRoomAvailabilityRequest" />
    <output message="s0: UpdateRoomAvailabilityResult" />
  </operation>
  ... other B2B hotel info update operations
</portType>
<message name="QueryRoomAvailabilityRequest">
  <part name="queryInfo" type="s0:RoomQuery" />
</message>
<message name="QueryRoomAvailabilityResult">
  <part name="availableRooms" type="s0:RoomAvailabilityArray" />
</message>
<portType name="GuestQueryHttpGet">
  <operation name="QueryRoomAvailabilty">
    <input message="s0:QueryRoomAvailabilityRequest"/>
    <output message="s0: QueryRoomAvailabilityResult" />
  </operation>
  ... other B2C guest query operations
</portType>
```

XML Test Harness



Building the Test Harness

Connecting to each of the arrows

Database interface

Message interface

Client API interface

Client GUI interface

Configuration / Logging interfaces

The crucial self-fulfilling prophecy...

Database Testing

DB usually runs in a separate process

Reduces risk of some integration bugs

Do high coverage unit testing of DB component

Functionality, performance, and volume tests

Does XML fit here ?

Integration tests focus on robustness

Verify application can handle broken DB connection, etc.

Exploit DB-to-XML connection for integration/system test

Initialize state by updating the DB from XML

Verify results by inspecting the DB with XML

Use SQL in combination with XPath

Message Testing

Messages should be based on specs

Session semantics are often simple

Pub/Sub environments are ideal for testing

Plumbing issues should be verified by unit tests

Payloads can be:

- XML

- Object graph or tree

- EDI

Convert to/from XML using JSX, JAXB, etc.

Client API Testing

Browser Based

XML payloads – rendered by client

Formatted payloads (HTML/WML)

Binary Payloads

SOAP based

All XML (except some attachments)

Structured (RMI/CORBA)

XML payloads

Other payloads

Client GUI testing

Minimize by testing more at the API level

Apply XML selectively

HTML interface is really an API

Integrate existing “record+script” tools

Generate their scripts from XSL

Use XML for special testing tasks

Internationalization

Customization

Usability testing can't be automated !

Test data management

Test Data Generation

Use XSLT to generate message sequences

Use “templates and variables”

Test results capture

Dump object graphs and convert offline

Capturing production data

Useful, but no substitute for specifications

Archiving, displaying, and reusing test data

Integrate with your bugtracking system

Data generation example

```
<GenerateRoomQuery>  
  <!-- Stay begins 21 days from when the test is run,  
        and lasts for two days. We don't care about  
        other fields. -->  
  <StayBegins>21</StayBegins>  
  <StayLength>2</StayLength>  
</GenerateRoomQuery>
```

XSLT Stylesheet

```
<RoomQuery>  
  <CheckinDate>2001-OCT-14</CheckinDate>  
  <CheckoutDate>2001-OCT-16</CheckoutDate>  
  <MinimumHotelRating>**</MinimumHotelRating>  
  <Smoking>YES</Smoking>  
  <MaximumTotalPrice>150.00</MaximumTotalPrice>  
</RoomQuery>
```

Test result validation

The key: use constraint programming !

Xpath /XSLT

Sufficient for simple validation

Easy to call out to Java (or whatever) for complex comparisons, e.g. of date values

XSLTUnit.org is a good example

SQL

Useful for white box verification of persistence

Can be combined with XPath

Full Logic programming

Ilog Jrules

Xpath-logic/LoPix

Result validation example

INPUT

```
<RoomQuery>
  <CheckinDate>2001-OCT-14</CheckinDate>
  <CheckoutDate>2001-OCT-16</CheckoutDate>
  <MinimumHotelRating>**</MinimumHotelRating>
  <Smoking>YES</Smoking>
  <MaximumTotalPrice>150.00</MaximumTotalPrice>
</RoomQuery>
```

OUTPUT

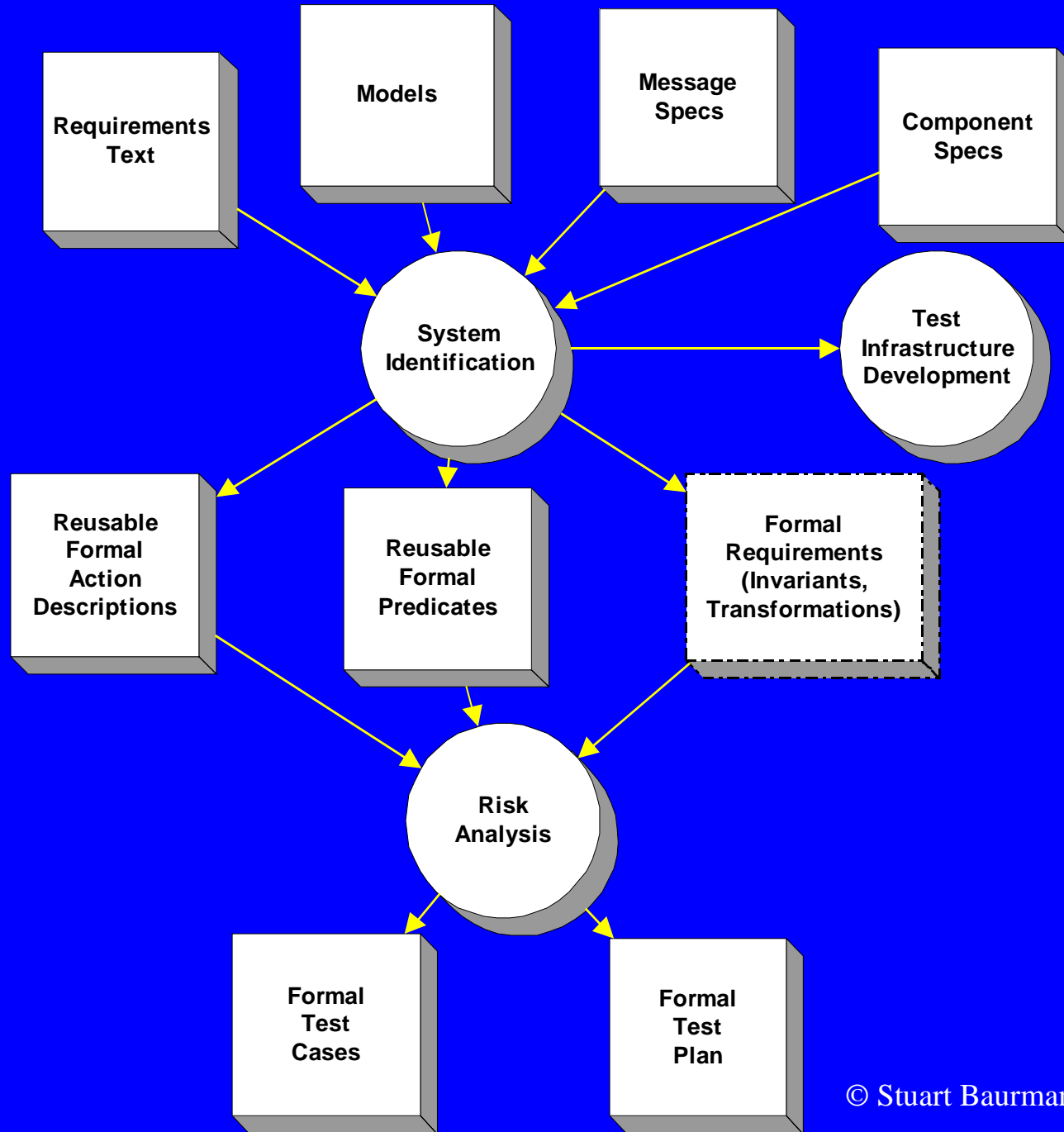
```
<RoomQuoteList>
  <RoomQuote ID="RQRAD839849234">
    <HotelName>Radisson Austin Arboreteum</HotelName>
    <Rating>***</Rating>
    <RoomDescription>STD 2 Queen, NSMK</RoomDescription>
    <Smoking>NO</Smoking>
    <CheckinDate>2001-OCT-14</CheckinDate>
    <CheckoutDate>2001-OCT-16</CheckoutDate>
    <TotalPrice>138.00</TotalPrice>
  </RoomQuote>
  <RoomQuote>
    ...
```

Validation example in XSL

Assuming I have a variable \$query with the RoomQuery in it:

```
<xsl:template match="RoomQuote">
  <xsl:if test="not (startsWith(Rating , $query/MinimumHotelRating))">
    <Error>Rating is too low for <xsl:copy-of select="."></Error>
  </xsl:if>
  <xsl:if test="Smoking != $query/Smoking">
    <Error>Smoking value incorrect for <xsl:copy-of
select="."></Error>
  </xsl:if>
  <xsl:if test="TotalPrice > $query/MaximumTotalPrice">
    <Error>Price too high for <xsl:copy-of select="."></Error>
  </xsl:if>
  <!-- Probably call out to some java functions to compare the dates.... -->
</xsl:template>
```

Creating Formal Test Plans



A Simple Test Case Format

```
<TestCase ID="TC_0000001">
  <Description>Verify RoomQuote results for a typical query</Description>
  <TargetModule type="SOAP_RPC_CALL">
    <WSDL_URL>http://www.hotelportal.com/rsv_svc.wsdl</WSDL_URL>
    <operationName>QueryRoomAvailability</operationName>
    ... in a non-SOAP context, this would be some other description of how
    to connect to the harness, submit the input, and retrieve the output.
  </TargetModule>
  <Input>
    <GeneratorStylesheetURL>roomQueryGen.xsl</GeneratorStylesheetURL>
    <GeneratorInputData>
      <GenerateRoomQuery>
        <StayBegins>21</StayBegins>
        <StayLength>2</StayLength>
      </GenerateRoomQuery>
    </GeneratorInputData>
  </Input>
  <ValidatorStylesheetURL>theValidationStylesheet.xsl</ValidatorStylesheetURL>
</TestCase>
```

Putting the Pieces together

- XML test plans and results are managed in a repository, alongside captured production data.
- Tests are executed using simple harnesses – results are captured into the repository.
- Results are validated using logic programs embedded (or referenced) in the test plan.
- All details of test execution remain viewable for comparison with future results.
- Test plans are versioned and evolved in an orderly manner.
- Testing facilities are exposed as web services in their own right.

Impact on the Software Development Process

(From Requirements to Testing.....and Back Again)

Generating test plans from requirements

Theory vs. practice

How are the detailed requirements really defined ?

Email/Conference-call negotiations during coding and testing

Ad hoc change requests (Extreme Programming indeed!)

Example: eXtreme B2B

- *January*
 - System requirements and functional spec docs are looking good
 - First implementation is working pretty well, meets requirements
 - Internal system testing, based on requirements spec, goes well
 - Marketing goes into full swing
 - Hotel Chain A signs up !
- *February*
 - Hotel Chain A specs arrive
 - No major changes required – a few small change requests are implemented quickly
 - Hotel Chain B signs up
 - 2 new, complex integration requirements
 - Complete specs are not yet in
 - Some architectural enhancements are needed
 - Change Requests created by analysts and developers

...B2B fiasco cont.

March

- Hotel A integration acceptance testing
 - A few misunderstandings are discovered
 - A few more change requests
 - Decision to deploy live anyway – first revenue generated !
 - Maintenance branch created
- Hotel B specs continue to trickle in
 - Development proceeds on trunk
 - February CRs are implemented and tweaked along the way
 - Unit tests also implemented, but system test plans languish

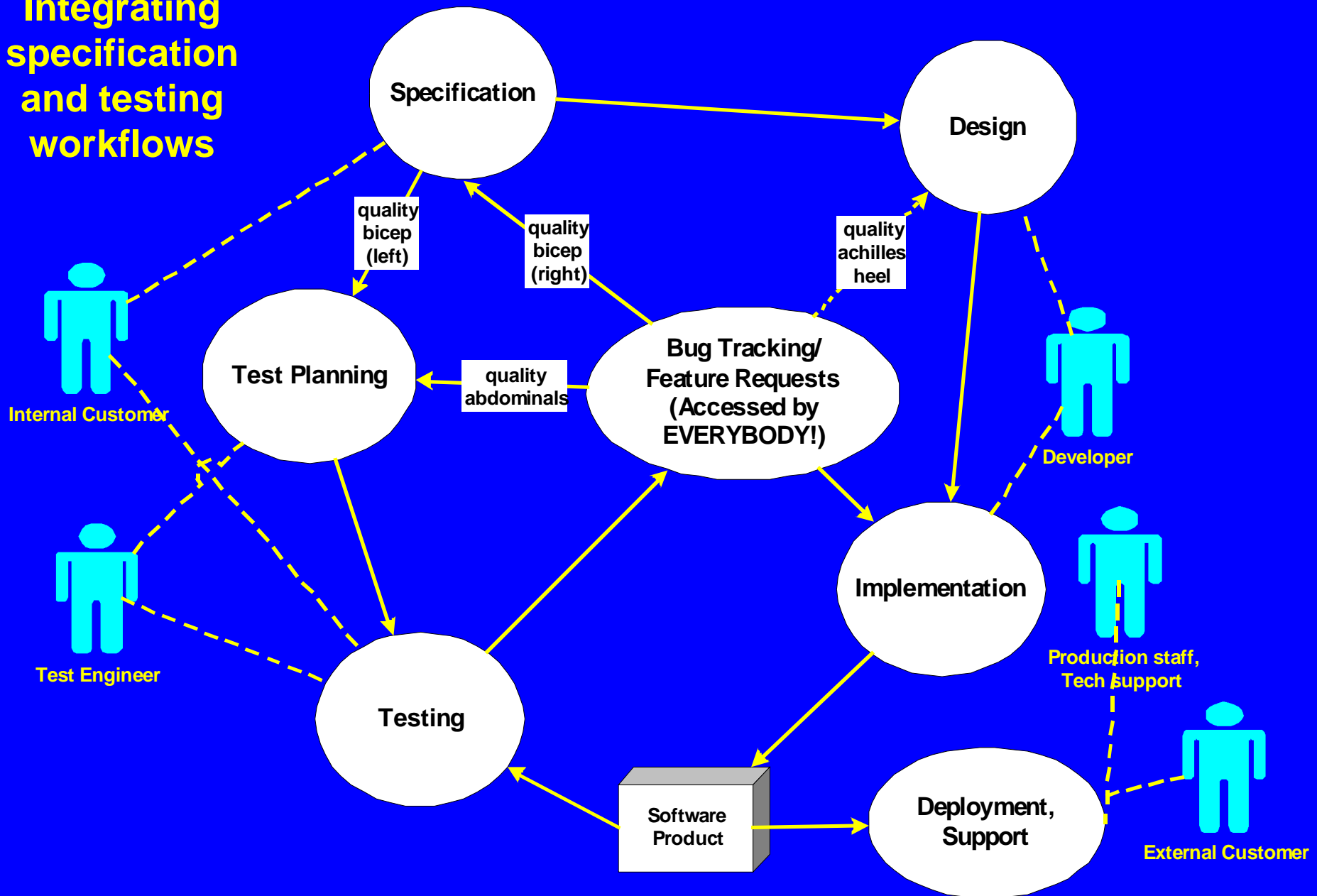
April

1. Hotel A continued success
 - Desirable new features identified
 - Some overlap with the Hotel B features
 - Hotel B internal testing
 - Coverage and efficiency are mediocre
 - Several bugs related to ambiguity of the spec are resolved on the spot

Specification and Test Plan Practices

- **System Test Plans should**
 - *refer to snapshots of a complete specification*
 - *itemize tested requirements (traceability)*
- **Change Requests should**
 - *always trigger specification updates*
- **Reported Defects (bugs) should**
 - *trigger spec updates for all changes or clarifications of specs required to resolve the defect*

Integrating specification and testing workflows



Conclusion

- Building XML interfaces to a system makes the system more readily and repeatably testable
- Many opportunities exist to automate test execution, and provide efficient test management environments
- Integrating the specification workflow with the testing workflow leads to more accurate specifications
- XML technology offers a good integration and content management platform for these efforts

Bottom Line Impact of XML:

Better Testing + Better Specifications

= Higher Software Quality !