Architecture of Enterprise Applications VIII
Testability and Usability

Haopeng Chen

RE*liable,* ***IN****telligent and* ***S****calable Systems Group (****REINS****)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

TESTABILITY

- Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.

- For a system to be properly testable, it must be possible to control each component's internal state and inputs and then to observe its outputs.

TESTABILITY

- Testing is done by various developers, testers, verifiers, or users and is the last step of various parts of the software life cycle.
- Portions of the code, the design, or the complete system may be tested.
- The response measures for testability deal with
  - how effective the tests are in discovering faults
  - and how long it takes to perform the tests to some desired level of coverage.

TESTABILITY

- Source of stimulus.
  - The testing is performed by unit testers, integration testers, system testers, or the client. A test of the design may be performed by other developers or by an external group.
- Stimulus.
  - The stimulus for the testing is that a milestone in the development process is met.
- Artifact.
  - A design, a piece of code, or the whole system is the artifact being tested.
- Environment.
  - The test can happen at design time, at development time, at compile time, or at deployment time.

TESTABILITY

- Response.
  - Since testability is related to observability and controllability, the desired response is that the system can be controlled to perform the desired tests and that the response to each test can be observed.

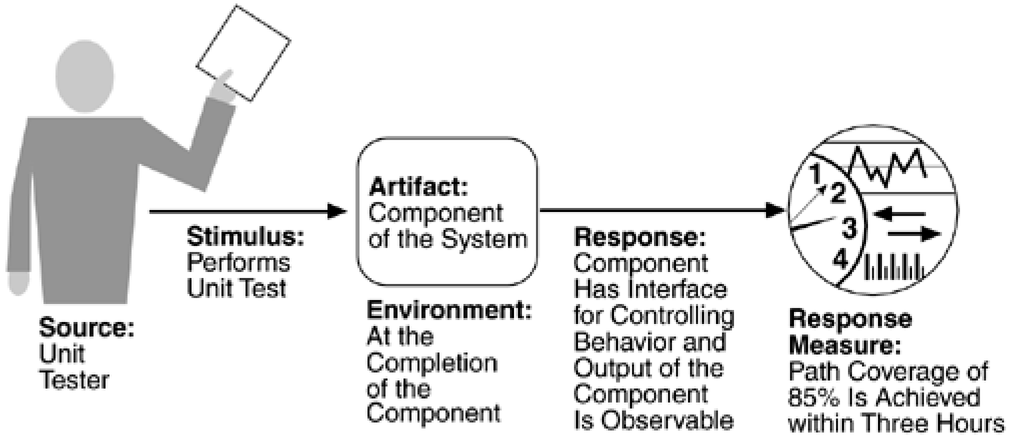
- Response measure.
  - Response measures are
    - the percentage of statements that have been executed in some test,
    - the length of the longest test chain (a measure of the difficulty of performing the tests),
    - and estimates of the probability of finding additional faults.

TESTABILITY

Portion of Scenario	Possible Values
Source	Unit developer Increment integrator System verifier Client acceptance tester System user
Stimulus	Analysis, architecture, design, class, subsystem integration completed; system delivered
Artifact	Piece of design, piece of code, complete application
Environment	At design time, at development time, at compile time, at deployment time
Response	Provides access to state values; provides computed values; prepares test environment
Response Measure	Percent executable statements executed Probability of failure if fault exists Time to perform tests Length of longest dependency chain in a test Length of time to prepare test environment

TESTABILITY

REliable, INtelligent & Scalable Systems

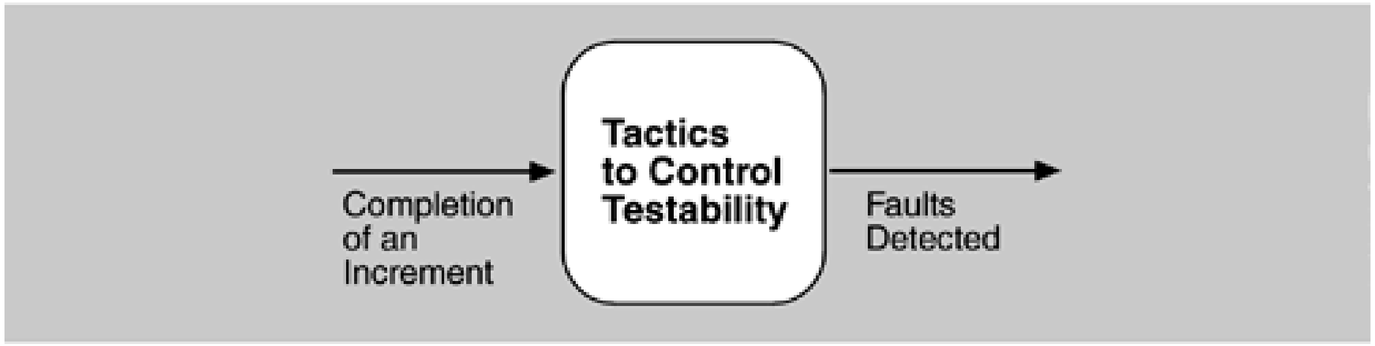


Testability Tactics

- The goal of tactics for testability is to allow for easier testing when an increment of software development is completed.

- Architectural techniques for enhancing the software testability have not received as much attention as more mature fields such as modifiability, performance, and availability,
- but, since testing consumes such a high percentage of system development cost, anything the architect can do to reduce this cost will yield a significant benefit.

Goal of testability tactics



Testability Tactics

- Although we included design reviews as a testing technique, we are concerned only with testing a running system.
  - The goal of a testing regimen is to discover faults. This requires that input be provided to the software being tested and that the output be captured.
- Executing the test procedures requires some software to provide input to the software being tested and to capture the output.
  - This is called a test harness.
  - A question we do not consider here is the design and generation of the test harness. In some systems, this takes substantial time and expense.
- We discuss two categories of tactics for testing: providing input and capturing output, and internal monitoring.

Testability Tactics-INPUT/OUTPUT

- There are three tactics for managing input and output for testing.
  - Record/playback.
    - Record/playback refers to both capturing information crossing an interface and using it as input into the test harness..
  - Separate interface from implementation.
    - Separating the interface from the implementation allows substitution of implementations for various testing purposes.
  - Specialize access routes/interfaces.
    - Having specialized testing interfaces allows the capturing or specification of variable values for a component through a test harness as well as independently from its normal execution.

Testability Tactics-INPUT/OUTPUT

- 在RUBiS中，其Client层实际上就采用了和录制/回放策略类似的策略，模拟指定数量的用户对RUBiS系统进行访问，以测试系统性能。它参数化的部分包括状态迁移的概率和用户思考时间等。

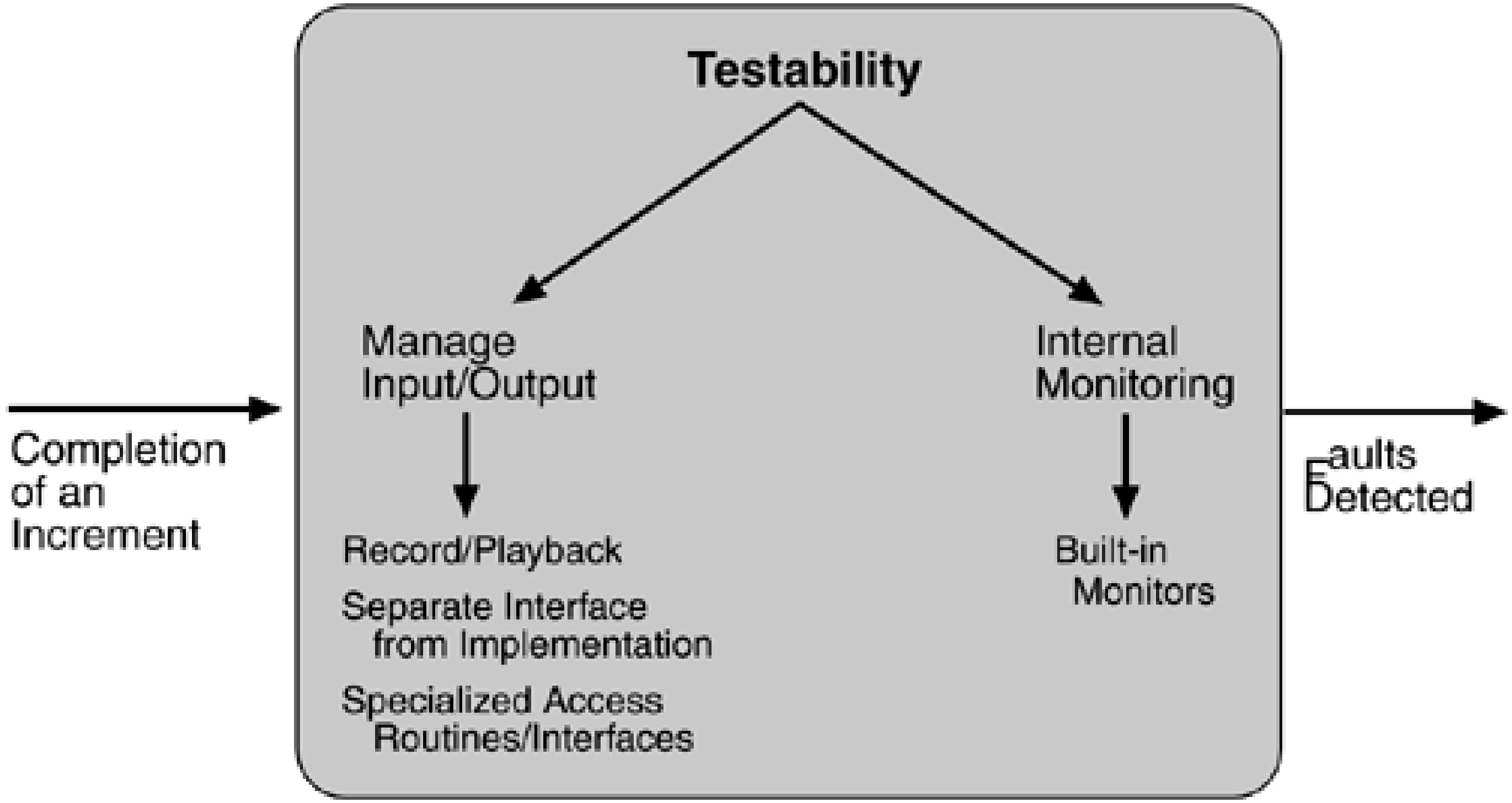
Testability Tactics-internal monitoring

- A component can implement tactics based on internal state to support the testing process.
  - Built-in monitors.
    - The component can maintain state, performance load, capacity, security, or other information accessible through an interface.

- 内部监视策略是通过内置监视器来实现的，其目的是要通过接口向外提供系统内部的状态。

- 例如，在很多设备中，都有开机自检功能，它会在加电时自检系统状态，然后以指示灯或屏幕显示的方式显示测试结果；还有的设备具有周期性自检功能，以及时发现运行中的设备出现的故障。内部监视通常会向系统提供一个标准输入，然后观察其输出是否符合预期值，而这个输入和输出对用户是透明的，并且不会对系统造成任何影响。

Testability Tactics-Summary



USABILITY

- Usability is concerned with
  - how easy it is for the user to accomplish a desired task and the kind of user support the system provides.

- It can be broken down into the following areas:
  - Learning system features.
  - Using a system efficiently.
  - Minimizing the impact of errors.
  - Adapting the system to user needs.
  - Increasing confidence and satisfaction.

USABILITY

- The normal development process detects usability problems through building prototypes and user testing.

- The later a problem is discovered and the deeper into the architecture its repair must be made, the more the repair is threatened by time and budget pressures.

USABILITY

- Source of stimulus.
  - The end user is always the source of the stimulus.
- Stimulus.
  - The stimulus is that the end user wishes to use a system efficiently, learn to use the system, minimize the impact of errors, adapt the system, or feel comfortable with the system.
- Artifact.
  - The artifact is always the system.
- Environment.
  - The user actions with which usability is concerned always occur at runtime or at system configuration time.

USABILITY

- Response.
  - The system should either provide the user with the features needed or anticipate the user's needs.
- Response measure.
  - The response is measured by task time, number of errors, number of problems solved, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, or amount of time/data lost when an error occurs.

USABILITY

Portion of Scenario	Possible Values
Source	End user
Stimulus	Wants to learn system features; use system efficiently; minimize impact of errors; adapt system; feel comfortable
Artifact	System
Environment	At runtime or configure time

USABILITY

REliable, INtelligent & Scalable Systems

Response	System provides one or more of the following responses:

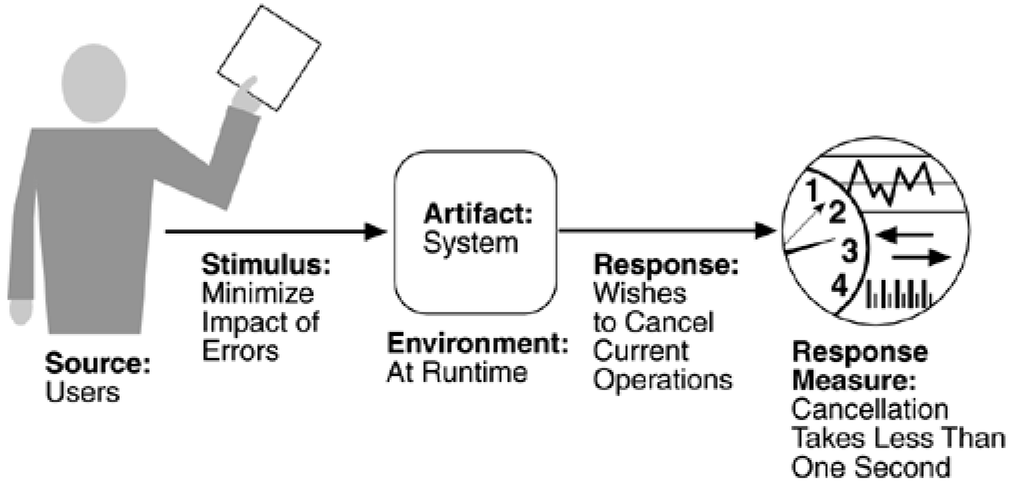
Response — **System provides one or more of the following responses:**

- to support "learn system features"
    - help system is sensitive to context;
    - interface is familiar to user;
    - interface is usable in an unfamiliar context
- to support "use system efficiently":
    - aggregation of data and/or commands;
    - re-use of already entered data and/or commands;
    - support for efficient navigation within a screen;
    - distinct views with consistent operations;
    - comprehensive searching;
    - multiple simultaneous activities
- to "minimize impact of errors":
    - undo, cancel, recover from system failure, recognize and correct user error, retrieve forgotten password, verify system resources
- to "adapt system":
    - customizability; internationalization
- to "feel comfortable":
    - display system state; work at the user's pace

USABILITY

REliable, INtelligent & Scalable Systems

Response Measure	**Task time, number of errors, number of problems solved, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, amount of time/data lost**

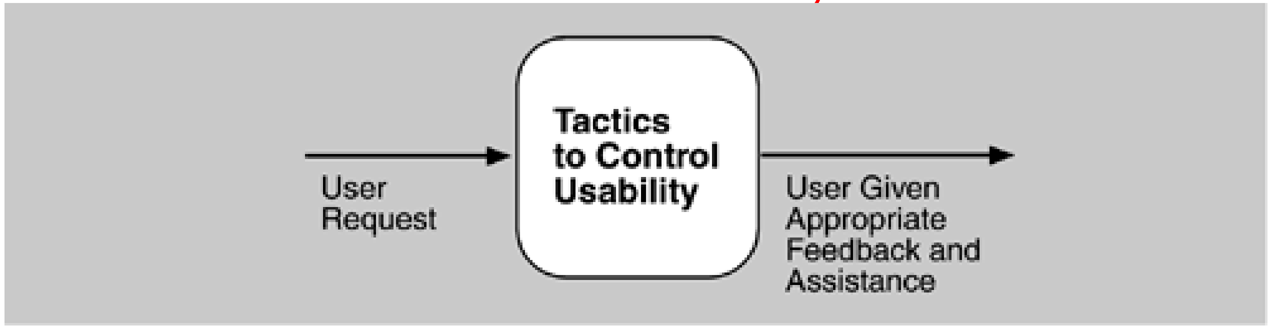

Source: Users
Stimulus: Minimize Impact of Errors
Artifact: System
Environment: At Runtime
Response: Wishes to Cancel Current Operations
1
2
3
4
Response Measure: Cancellation Takes Less Than One Second

Usability Tactics

- Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of support the system provides to the user.

- Two types of tactics support usability, each intended for two categories "users".
  - The first category, Runtime, includes those that support the user during system execution.
  - The second category is based on the iterative nature of user interface design and supports the interface developer at design time.

Goal of runtime usability tactics


User Request
Tactics to Control Usability
User Given Appropriate Feedback and Assistance

Usability Tactics-Runtime tactics

- Once a system is executing, usability is enhanced
  - by giving the user feedback as to what the system is doing
  - and by providing the user with the ability to issue usability-based commands
    - such as those we saw. For example, cancel, undo, aggregate, and show multiple views support the user in either error correction or more efficient operations.

  - Maintain a model of the task.
    - The task model is used to determine context so the system can have some idea of what the user is attempting and provide various kinds of assistance.
    - For example, knowing that sentences usually start with capital letters would allow an application to correct a lower-case letter in that position.

Usability Tactics-Runtime tactics

- Maintain a model of the user.
  - In this case, the model maintained is of the user.
  - It determines the user's knowledge of the system, the user's behavior in terms of expected response time, and other aspects specific to a user or a class of users.
  - For example, maintaining a user model allows the system to pace scrolling so that pages do not fly past faster than they can be read.

- Maintain a model of the system.
  - In this case, the model maintained is that of the system.
  - It determines the expected system behavior so that appropriate feedback can be given to the user.
  - The system model predicts items such as the time needed to complete current activity.

Usability Tactics- design time tactics

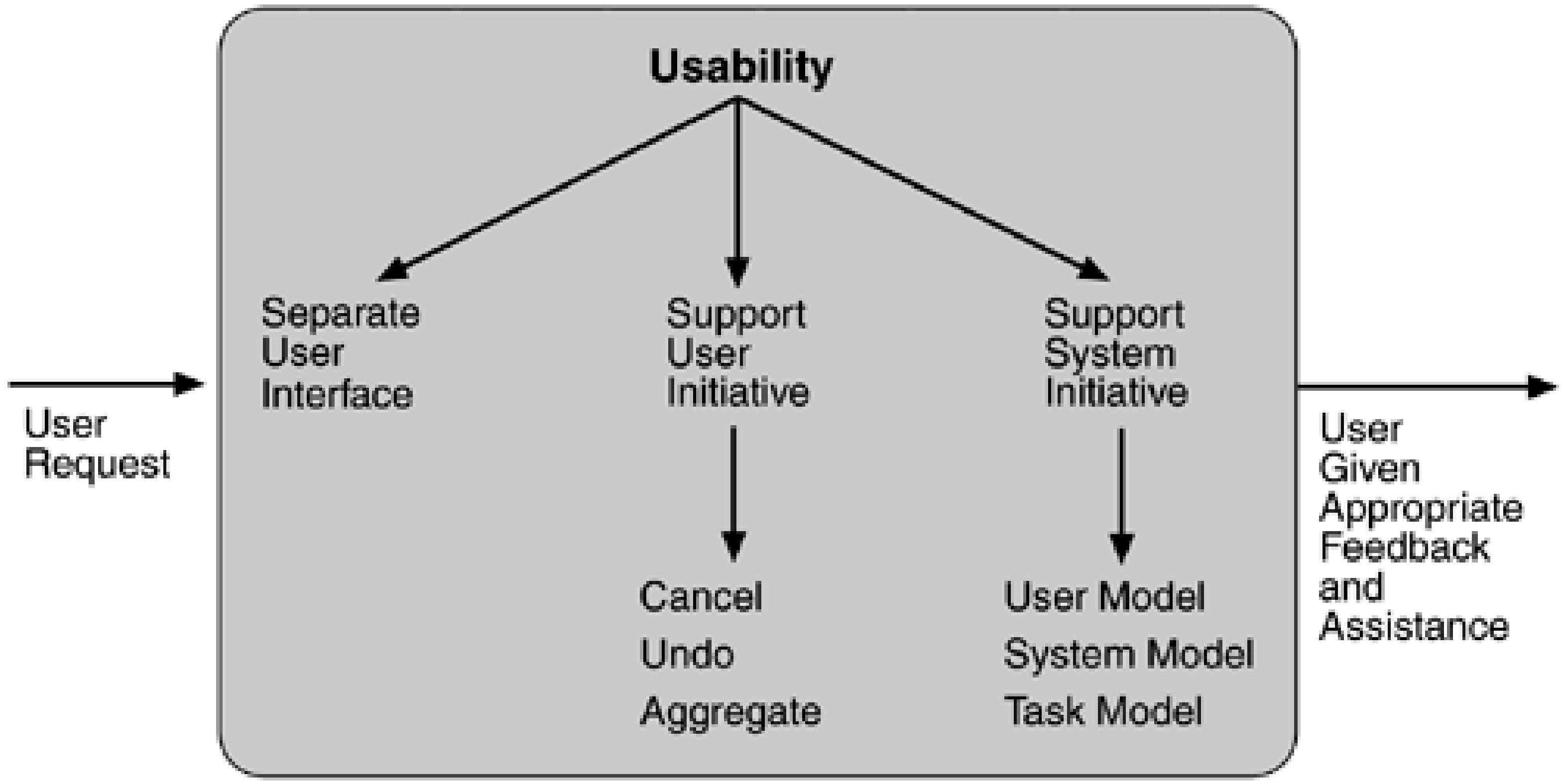
- User interfaces are typically revised frequently during the testing process.
  - That is, the usability engineer will give the developers revisions to the current user interface design and the developers will implement them.

- This leads to a tactic that is a refinement of the modifiability tactic of semantic coherence:
  - Separate the user interface from the rest of the application.
    - Model-View-Controller
    - Presentation-Abstraction-Control

Usability Tactics- design time tactics

- RUBiS实际上并未涉及易用性方面的内容，因为易用性并不在它的考虑范围内。
- 但是我们可以举另外一个例子，例如Microsoft Office套件的软件，例如office，就有undo和redo这样的功能，这就是易用性方面的典型例子。
- 看起来这个功能似乎很琐碎，很难认为其是架构设计层面需要考虑的问题，但其实并非如此。
  - 当undo和redo可以支持很多步时，就会在系统中占用许多内存资源来缓存各个中间版本，这就需要我们对缓存内容的数据结构和存储方式进行架构层面的设计了。因此，易用性有些是一些关注于用户体验的细节，但是有些则是关系到软件架构层面的问题，我们需要认真分析，区别对待。

Usability Tactics-Summary



Quality Attribute Stimuli

Availability	Unexpected event, nonoccurrence of expected event
Modifiability	Request to add/delete/change/vary functionality, platform, quality attribute, or capacity
Performance	Periodic, stochastic, or sporadic
Security	Tries to display, modify, change/delete information, access, or reduce availability to system services
Testability	Completion of phase of system development
Usability	Wants to learn system features, use a system efficiently, minimize the impact of errors, adapt the system, feel comfortable

Other System Quality Attributes

- A number of other attributes can be found in the attribute taxonomies in the research literature and in standard software engineering textbooks, and we have captured many of these in our scenarios.
  - For example, scalability is often an important attribute, but in our discussion here scalability is captured by modifying system capacity—the number of users supported, for example. Portability is captured as a platform modification.
- If some quality attribute—say interoperability—is important to your organization, it is reasonable to create your own general scenario for it.
  - For interoperability, a stimulus might be a request to interoperate with another system, a response might be a new interface or set of interfaces for the interoperation, and a response measure might be the difficulty in terms of time, the number of interfaces to be modified, and so forth.

Business Qualities

- In addition to the qualities that apply directly to a system, a number of business quality goals frequently shape a system's architecture. These goals center on cost, schedule, market, and marketing considerations.

  – Time to market.
    - If there is competitive pressure or a short window of opportunity for a system or product, development time becomes important.

  – Cost and benefit.
    - The development effort will naturally have a budget that must not be exceeded.

  – Projected lifetime of the system.
    - If the system is intended to have a long lifetime, modifiability, scalability, and portability become important.

Business Qualities

- Targeted market.
  - For general-purpose (mass-market) software, the platforms on which a system runs as well as its feature set will determine the size of the potential market.
- Rollout schedule.
  - If a product is to be introduced as base functionality with many features released later, the flexibility and customizability of the architecture are important.
- Integration with legacy systems.
  - If the new system has to integrate with existing systems, care must be taken to define appropriate integration mechanisms.

Architecture Qualities

- Conceptual integrity is the underlying theme or vision that unifies the design of the system at all levels.

- Correctness and completeness are essential for the architecture to allow for all of the system's requirements and runtime resource constraints to be met.

- Buildability allows the system to be completed by the available team in a timely manner and to be open to certain changes as development progresses.

Relationship of Tactics to Architectural Patterns

REliable, INtelligent & Scalable Systems

- In fact, an architect usually chooses a pattern or a collection of patterns designed to realize one or more tactics. However, each pattern implements multiple tactics, whether desired or not.

  The Active Object design pattern decouples method execution from method invocation to enhance concurrency and simplify synchronized access to objects that reside in their own thread of control.

- The pattern consists of six elements:
  - a proxy, which provides an interface that allows clients to invoke publicly accessible methods on an active object;
  - a method request, which defines an interface for executing the methods of an active object;
  - an activation list, which maintains a buffer of pending method requests;
  - a scheduler, which decides what method requests to execute next;
  - a servant, which defines the behavior and state modeled as an active object;
  - and a future, which allows the client to obtain the result of the method invocation.

Relationship of Tactics to Architectural Patterns

- The motivation for this pattern is to enhance concurrency—a performance goal. Thus, its main purpose is to implement the "introduce concurrency" performance tactic.
  - Information hiding (modifiability). Each element chooses the responsibilities it will achieve and hides their achievement behind an interface.
  - Intermediary (modifiability). The proxy acts as an intermediary that will buffer changes to the method invocation.
  - Binding time (modifiability). The active object pattern assumes that requests for the object arrive at the object at runtime. The binding of the client to the proxy, however, is left open in terms of binding time.
  - Scheduling policy (performance). The scheduler implements some scheduling policy.

Relationship of Tactics to Architectural Patterns

- Any pattern implements several tactics, often concerned with different quality attributes, and any implementation of the pattern also makes choices about tactics.
  - For example, an implementation could maintain a log of requests to the active object for supporting recovery, maintaining an audit trail, or supporting testability.
- The analysis process for the architect involves understanding all of the tactics embedded in an implementation, and the design process involves making a judicious choice of what combination of tactics will achieve the system's desired goals.

References

- Software.Architecture.In.Practice.2nd.Edition

Thank You!