Software Architecture and Design Overview II

Mark C. Paulk, Ph.D.

Mark.Paulk@utdallas.edu*, *Mark.Paulk@ieee.org
http://mark.paulk123.com/

Software Architecture Topics

Introduction to Architecture

Quality Attributes

- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

What Is An "Agile Method"?

A software engineering "methodology" that follows the Agile Manifesto?

A method that supports responding rapidly to changing requirements?

- **Mark Paulk**

Does an agile method necessarily imply

- **Evolutionary / iterative / incremental development?**
- **Empowerment / participation of the development team?**
- **Active collaboration with the customer?**
- **…**

Agile Manifesto

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Agile Principles

Customer satisfaction by early and continuous delivery of valuable software

Welcome changing requirements, even late in development

Deliver working software frequently

Business people and developers work together daily

Build projects around motivated individuals

Agile Principles -1

Face-to-face conversation is the most effective and efficient method of conveying information

Working software is the primary measure of progress

Promote sustainable development

Able to maintain a constant pace indefinitely

Close, daily co-operation between business people and developers

Agile Principles -2

Continuous attention to technical excellence and good design

Simplicity

Self-organizing teams

Reflect on how to become more effective, tune and adjust behavior

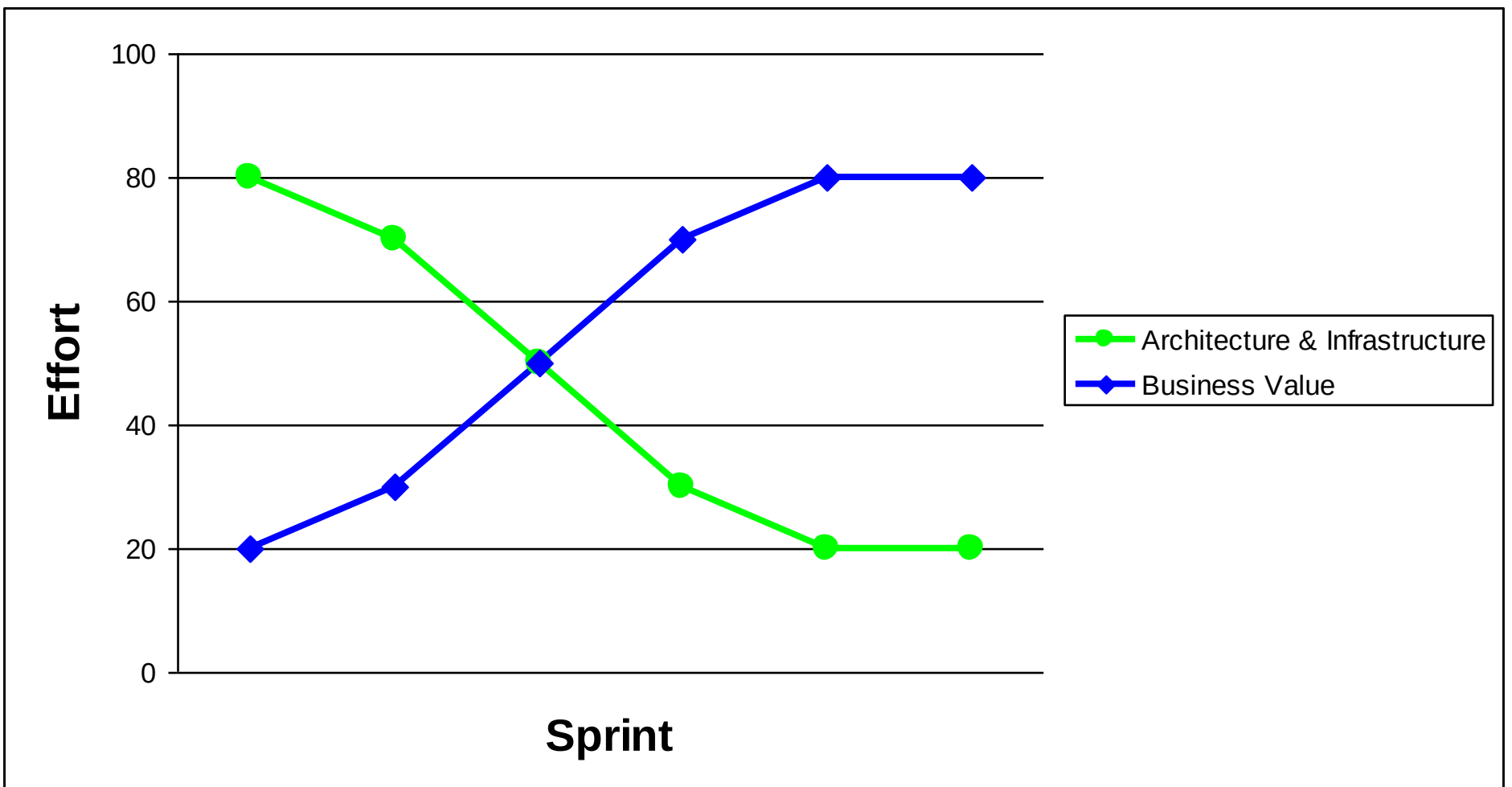
Architecture in an Agile Context

The best teams may be self-organizing, but the best architectures still require technical skill, deep experience, and deep knowledge.

A focus on early and continuous release of software, where "working" is measured in terms of customer-facing features, leaves little time for addressing the kinds of cross-cutting concerns and infrastructure critical to a high-quality large-scale system.

The issue is not agile vs architecture but how to best blend agile and architecture…

Building the Foundation



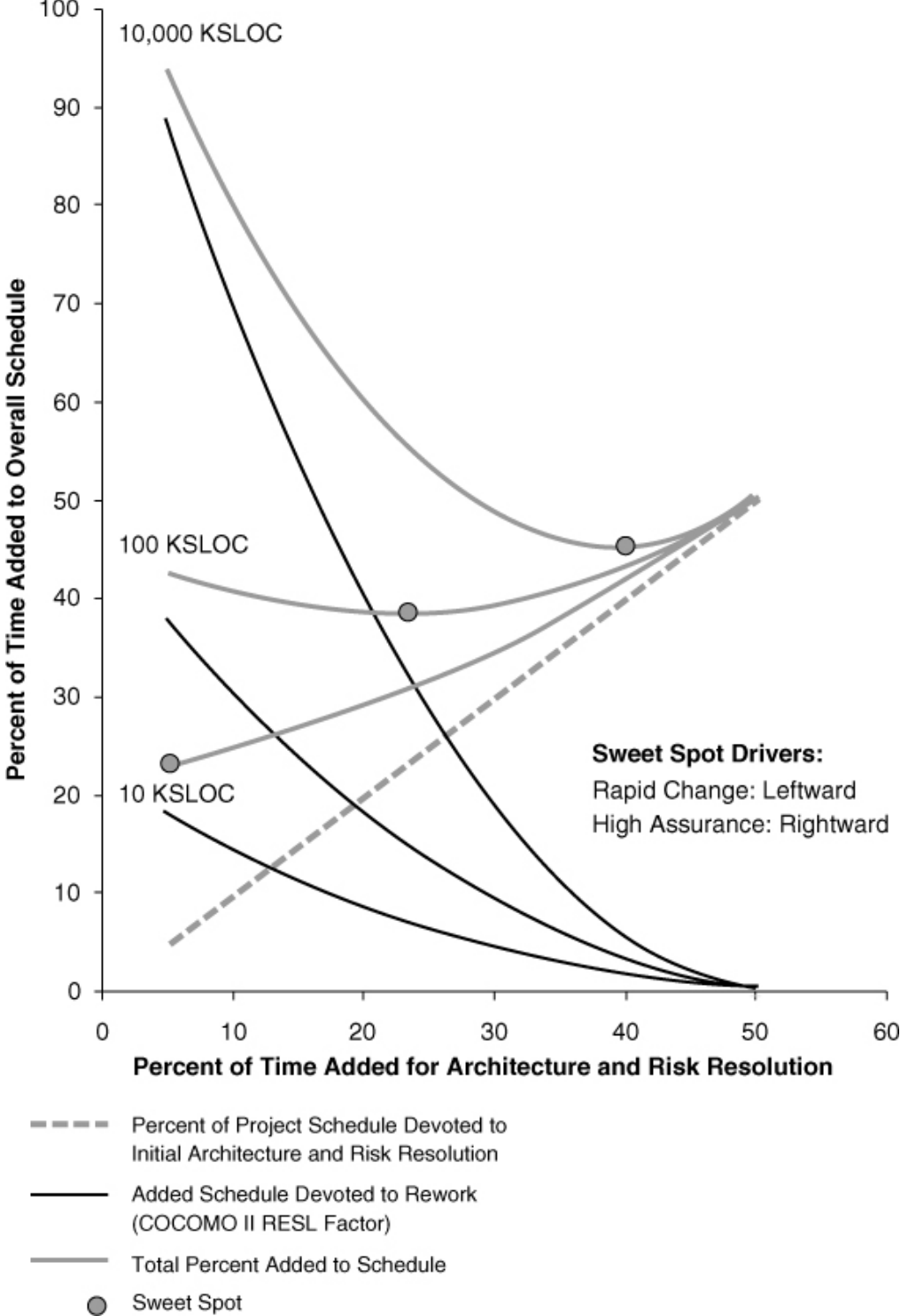
Up-Front Work vs Agility

Boehm and Turner analyzed the effects of up-front architecture and risk resolution effort. (COCOMO II RESL)

Up-front design work on the architecture vs rework

Amount of architecture and risk resolution effort is plotted as the dashed line, moving up and to the right from near the origin

- **project of 10 KSLOC**
- **project of 100 KSLOC**
- **project of 1,000 KSLOC**


10,000 KSLOC
100 KSLOC
10 KSLOC
Percent of Time Added to Overall Schedule
Percent of Time Added for Architecture and Risk Resolution
0
10
20
30
40
50
60
70
80
90
100
Sweet Spot Drivers:
Rapid Change: Leftward
High Assurance: Rightward
Percent of Project Schedule Devoted to Initial Architecture and Risk Resolution
Added Schedule Devoted to Rework (COCOMO II RESL Factor)
Total Percent Added to Schedule
Sweet Spot

Architecture vs Agility Tradeoff

Adding time for up-front work reduces later rework.

There is a sweet spot for each project.

- **10 KSLOC project, at the far left**
- **100 KSLOC project, around 20% of the project schedule**
- **1,000 KSLOC project, around 40% of the project schedule**

No one answer is appropriate for all situations.

Documentation and YAGNI

Expect the greatest agile friction from evaluation and documentation.

Technical documentation principle: write for the reader.

- **No reader → no documentation**

The Views and Beyond approach

- **uses the architectural view as the "unit" of documentation**
- **prescribes producing a view if and only if it addresses substantial concerns of an important stakeholder community**
- **the view selection method prescribes producing the documentation in prioritized stages to satisfy the needs of the stakeholders who need it now**

Incremental Commitment Model
(Boehm)

Commitment and accountability of success-critical stakeholders

Stakeholder "satisficing"

Incremental and evolutionary growth of system definition and stakeholder commitment

Iterative system development and definition

Interleaved system definition and development

Risk management

Guidelines for Agile Architecture
(Booch)

All good software-intensive architectures are agile.

- **a successful architecture is resilient and loosely coupled**
- **composed of a core set of well-reasoned design decisions**
- **contains some "wiggle room" that allows modifications to be made and refactorings to be done**

An effective agile process will allow the architecture to grow incrementally as the system is developed and matures.

- **decomposability**
- **separation of concerns**
- **near-independence of the parts**

The architecture should be visible and self-evident in the code

- **make the design patterns, cross-cutting concerns, and other important decisions obvious, well communicated, and defended**
- **may, in turn, require documentation**
- **"socialize" the architecture**

Technical Debt

Ward Cunningham first drew the comparison between technical complexity and debt in 1992.

Shipping first time code is like going into debt.

- **A little debt speeds development so long as it is paid back promptly with a rewrite...**
- **The danger occurs when the debt is not repaid.**
- **Every minute spent on not-quite-right code counts as interest on that debt.**

Activities that might be postponed include

- **documentation**
- **writing tests**
- **attending to to-do comments**
- **tackling compiler and static code analysis warnings**
- **knowledge that isn't shared around the organization**
- **code that is too confusing to be modified easily**

Tradeoff Advice

Large and complex system with relatively stable and well-understood requirements

- **do a large amount of architecture work up front**

Big projects with vague or unstable requirements

- **quickly design a complete candidate architecture**
- **Cockburn's Crystal Clear "walking skeleton"**

Smaller projects with uncertain requirements,

- **try to get agreement on the central patterns**

Software Architecture Topics

Introduction to Architecture

Quality Attributes

- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Architecturally Significant Requirements (ASRs)

Requirements documents

- **most of what is in a requirements specification does not affect the architecture**
- **much of what is useful to an architect is not in even the best requirements document**
- **ASRs often derive from business goals in the development organization**
- **excavation and archaeology is required to dig ASRs from requirements documents**

Design Decision Category	Look for Requirements Addressing . . .
Allocation of Responsibilities	Planned evolution of responsibilities, user roles, system modes, major processing steps, commercial packages
Coordination Model	Properties of the coordination (timeliness, currency, completeness, correctness, and consistency) Names of external elements, protocols, sensors or actuators (devices), middleware, network configurations (including their security properties) Evolution requirements on the list above
Data Model	Processing steps, information flows, major domain entities, access rights, persistence, evolution requirements
Management of Resources	Time, concurrency, memory footprint, scheduling, multiple users, multiple activities, devices, energy usage, soft resources (buffers, queues, etc.) Scalability requirements on the list above
Mapping among Architectural Elements	Plans for teaming, processors, families of processors, evolution of processors, network configurations
Binding Time Decisions	Extension of or flexibility of functionality, regional distinctions, language distinctions, portability, calibrations, configurations
Choice of Technology	Named technologies, changes to technologies (planned and unplanned)

Interviewing Stakeholders

Architects often have good ideas what quality attributes are exhibited by similar systems and are reasonable.

Stakeholders often have no idea what quality attributes they want in a system.

Results of stakeholder interviews

- **a list of architectural drivers**
- **a set of quality attribute scenarios that the stakeholders (as a group) prioritized**

Quality Attribute Workshop

1) **QAW Presentation and Introductions**
2) **Business/Mission Presentation**
3) **Architectural Plan Presentation**
4) **Identification of Architectural Drivers**
5) **Scenario Brainstorming**
6) **Scenario Consolidation**
7) **Scenario Prioritization**
8) **Scenario Refinement**

Gathering ASRs by Understanding the Business Goals

Business goals are the reason for building a system.

- **often the precursor of requirements that may or may not be captured in a requirements specification**

Business goals often lead to quality attribute requirements.

- **every quality attribute requirement should originate from some higher purpose that can be described in terms of added value**

Business goals may directly affect the architecture without precipitating a quality attribute requirement at all.

Standard Business Goal Categories

1. Contributing to the growth and continuity of the organization
2. Meeting financial objectives
3. Meeting personal objectives
4. Meeting responsibility to employees
5. Meeting responsibility to society
6. Meeting responsibility to state
7. Meeting responsibility to shareholders
8. Managing market position
9. Improving business processes
10. Managing the quality and reputation of products
11. Managing change in environmental factors

Pedigreed Attribute eLicitation Method (PALM)

Day and a half workshop attended by architects and stakeholders who can speak to the business goals of the organizations involved

1) **PALM overview presentation**
2) **Business drivers presentation**
3) **Architecture drivers presentation**
4) **Business goals elicitation**
5) **Identification of potential quality attributes from business goals**
6) **Assignment of pedigree to existing quality attribute drivers**
7) **Exercise conclusion**

Utility Tree

Begins with the word “utility” as the root node.

List the major quality attributes that the system is required to exhibit.

- **under each quality attribute, record a specific refinement of that QA**
- **under each refinement, record the appropriate ASRs (usually expressed as QA scenarios)**

Evaluate against two criteria

- **the business value of the candidate ASR**
- **the architectural impact of including it**
  - **must-have, important, nice-to-have**

Tying the Methods Together

If you have a requirements process that gathers, identifies, and prioritizes ASRs, consider yourself lucky…

If nobody has captured the business goals behind the system you're building, then a PALM exercise.

If you feel that important stakeholders have been overlooked, capture their concerns through interviews.

- **Quality Attribute Workshop**

Building a utility tree is a good way to capture ASRs along with their prioritization.

Blending Methods

PALM makes an excellent "subroutine call" from a Quality Attribute Workshop for the step that asks about business goals.

A quality attribute utility tree makes an excellent repository for the scenarios that are the workshop's output.

Pick the approach that fills in the biggest gap in your existing requirements:

- **stakeholder representation**
- **business goal manifestation**
- **ASR prioritization**

Designing an Architecture

The building blocks for designing a software architecture:

- **locating architecturally significant requirements**
- **capturing quality attribute requirements**
- **choosing, generating, tailoring, and analyzing design decisions for achieving those requirements**

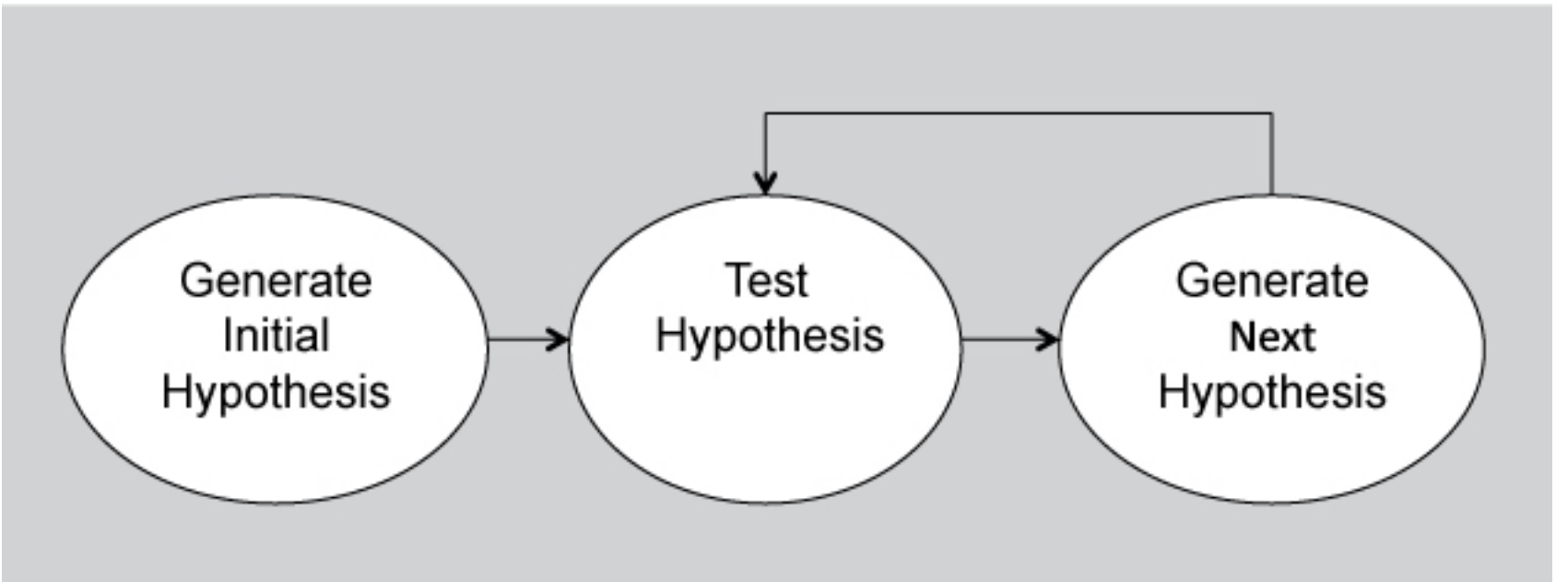
Now to pull the pieces together…

Design Strategy

Three ideas key to architecture design methods

- **decomposition**
  - **quality attributes refer to the system as a whole**
  - **as the design is decomposed, QA are too and assigned to elements of the decomposition**
- **designing to architecturally significant requirements**
  - **non-ASR requirements may not be met**
  - **1) relax the non-ASR requirement**
  - **2) re-prioritize and re-visit the design**
  - **3) don't meet the non-ASR requirement**
- **generate and test**
  - **testing determines whether the design meets the requirements**

Generate and Test



Generate and test as a design strategy leads to the following questions

1) **Where does the initial design hypothesis come from?**
2) **What are the tests that are applied?**
3) **How is the next hypothesis generated?**
4) **When are you done?**

Creating the Initial Hypothesis

Design solutions are created using "collateral" that is available to the project.

- **existing systems**
- **frameworks**
- **patterns and tactics**
- **domain decomposition**
- **design checklists**

Choosing the Tests

Three sources of tests

- **analysis techniques**
- **design checklists for quality attributes**
- **ASRs**

Generating the Next Hypothesis

If you have concerns… a list of quality attribute problems

Use design tactics to improve the design with respect to the particular quality attribute.

Terminating the Process

If you do not produce an acceptable design within budget…

1) **Proceed to implementation with the best hypothesis you were able to produce.**
   - **some ASRs may not be met and may need to be relaxed or eliminated**

2) **Argue for more budget for design and analysis.**
   - **revisit some of the major early design decisions**

3) **Suggest that the project be terminated.**

Attribute-Driven Design (ADD) Method

Produce a workable architecture quickly

Before beginning a design process, the requirements should (ideally) be known…

Requirements (changes) are continually arriving…

ADD can begin when a set of architecturally significant requirements is known.

ADD Inputs

ASRs

Context description

- **What are the boundaries of the system being designed?**
- **What are the external systems, devices, users, and environmental conditions with which the system being designed must interact?**

ADD Outputs

A set of sketches of architectural views

Module decomposition view

Other views according to the design solutions chosen

Breadth vs Depth First

Personnel availability may dictate a refinement strategy.

Risk mitigation may dictate a refinement strategy.

Deferral of some functionality or quality attribute concerns may dictate a mixed approach.

All else being equal, a breadth-first refinement strategy is preferred because

- **it allows you to apportion the most work to the most teams soonest**
- **allows for consideration of the interaction among the elements at the same level**

Generate a Design Solution

Sources of design candidates— patterns, tactics, and checklists

- **initial candidate design will likely be inspired by a pattern**
- **possibly augmented by one or more tactics**
- **consider the design checklists for the quality attributes**

To the extent that the system you're building is similar to others, it is likely that the solutions you choose will solve a collection of ASRs simultaneously…

Verify and Refine Requirements

Your design solution may not satisfy all the ASRs.

Backtrack – reconsider the design.

Unsatisfied ASRs may relate to

- **A quality attribute requirement allocated to the parent element**
- **A functional responsibility of the parent element**
- **One or more constraints on the parent element**

What Requirements Are Left?

Requirements assigned to element are satisfied…

Delegate to one of the children

Distribute among the children

Cannot be satisfied with the current design

- **backtrack**
- **push back on the requirement**

Done?

Terminate with a sketch of the architecture…

- **flesh out the architecture consistent with the overall design approaches laid out**

Satisfy (contractual) specifications…

Exhaust design budget…

Terminating ADD and releasing the architecture are different decisions.

- **early architectural views can be usable**

Software Architecture Topics

Introduction to Architecture

Quality Attributes

- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Documenting Software Architectures

If it is not written down, it does not exist.

- **Philippe Kruchten**

If you don’t have it in writing, I didn’t make a commitment.

- **mcp**

(A lack of planning on your part does not constitute a crisis on my part.)

- **mcp**

Architecture has to be communicated in a way to let its stakeholders use it properly to do their jobs.

Uses of Architecture Documentation

As a means of education

- **introducing people to the system**

As a primary vehicle for communication among stakeholders

- **including the architect in the project's future**

As the basis for system analysis and construction

Notations

Informal notations

- general-purpose diagramming and editing tools and visual conventions

Semiformal notations

- a standardized notation that prescribes graphical elements and rules of construction, e.g., UML

Formal notations

- has a precise (usually mathematically based) semantics
- formal analysis of both syntax and semantics is possible
- generally referred to as architecture description languages
- the use of such notations is rare

Views

A representation of a set of system elements and relations among them — not all system elements, but those of a particular type.

Let us divide the multidimensional entity that is a software architecture into a number of manageable representations of the system.

Documenting an architecture is a matter of documenting the relevant views and then adding documentation that applies to more than one view → Views and Beyond

Module Views

A module is an implementation unit that provides a coherent set of responsibilities.

The relations that modules have to one another include *is part of*, *depends on*, and *is a*.

It is unlikely that the documentation of any software architecture can be complete without at least one module view.

Component-and-Connector Views

Show elements that have some runtime presence

- **processes, objects, clients, servers, and data stores**

Include as elements the pathways of interaction

- **communication links and protocols, information flows, and access to shared storage**

Components have interfaces called ports.

Connectors have roles, which are its interfaces, defining the ways in which the connector may be used by components to carry out interaction.

Notations for C&C Views

Assign each component type and each connector type a separate visual form (symbol), and list each of the types in a key.

- **UML components are a good semantic match to C&C components.**
- **UML ports are a good semantic match to C&C ports.**
- **UML connectors cannot have substructure, attributes, or behavioral descriptions.**
  - **UML connectors are not always rich enough to represent C&C connectors**
  - **represent a "simple" C&C connector using a UML connector – a line**

Allocation Views

Describe the mapping of software units to elements of an environment in which the software is developed or in which it executes.

The relation in an allocation view is allocated to.

The usual goal of an allocation view is to compare

- **the properties required by the software element with**
- **the properties provided by the environmental elements**

to determine whether the allocation will be successful or not.

Quality Views

Module, C&C, and allocation views are all structural views.

In some systems structural views may not be the best way to present the architectural solution.

- **certain quality attributes are particularly important and pervasive**
- **the solution may be spread across multiple structures that are inconvenient to combine**

Extract the relevant pieces of the structural views and package them together.

Examples of Quality Views -1

Security view

- **can show all of the architectural measures taken to provide security**

Communications view

- **for systems that are globally dispersed and heterogeneous**
- **show all of the component-to-component channels, the various network channels, quality-of-service parameter values, and areas of concurrency**

Exception or error-handling view

- **illuminate and draw attention to error reporting and resolution mechanisms**

Examples of Quality Views -2

Reliability view

- **model reliability mechanisms such as replication and switchover a**
- **depict timing issues and transaction integrity**

Performance view

- **include those aspects of the architecture useful for inferring the system's performance**
- **network traffic models, maximum latencies for operations, and so forth**

Choosing the Views

At a minimum, expect to have at least one module view, at least one C&C view, and for larger systems, at least one allocation view in your architecture document.

A Three-Step Method for Choosing Views

Build a stakeholder/view table.

- **describe how much information the stakeholder requires from the view**

Combine views.

- **look for marginal views in the table: those that require only an overview, or that serve very few stakeholders**

Prioritize and stage

- **the decomposition view is a particularly helpful view to release early**
- **providing 80% of the information goes a long way,**
- **you don’t have to complete one view before starting another**

Combining Views

All views in an architecture are part of that same architecture and exist to achieve a common purpose.

Sometimes the most convenient way to show a strong association between two views is to collapse them into a single combined view.

- **can be very useful as long as you do not try to overload them with too many mappings**

Create an overlay that combines the information.

- **works well if the coupling between the two views is tight**

Building the Documentation Package -1

Section 1: The Primary Presentation

- **shows the elements and relations of the view**
- **most often graphical.**
- **lack of a key is the most common mistake that we see in documentation in practice.**

Section 2: The Element Catalog

- **elements and their properties**
- **relations and their properties**
- **element interfaces**
- **element behavior**

Section 3: Context Diagram

- **shows how the system or portion of the system depicted in this view relates to its environment**

Building the Documentation Package -2

Section 4: Variability Guide

- **shows how to exercise any variation points that are a part of the architecture shown in this view**

Section 5: Rationale

- **explains why the design reflected in the view came to be**
- **explain why the design is as it is and provide a convincing argument that it is sound**

Documenting Behavior

Traces – sequences of activities or interactions that describe the system's response to a specific stimulus when the system is in a specific state

- **use cases**
- **UML sequence diagram**
- **UML communication diagram**
- **UML activity diagram**

Comprehensive models show the complete behavior of structual elements

- **UML state machine diagram notation**
- **Architecture Analysis and Design Language (AADL)**
- **Specification and Description Language (SDL)**

Architecture Documentation and Quality Attributes -1

Any major design approach will have quality attribute properties associated with it.

- **client-server is good for scalability, layering is good for portability, …**
- **explaining the choice of approach is likely to include a discussion about the satisfaction of quality attribute requirements and tradeoffs incurred**

Individual architectural elements that provide a service often have quality attribute bounds assigned to them.

- **quality attribute bounds are defined in the interface documentation for the elements, sometimes in the form of a service-level agreement**

Architecture Documentation and Quality Attributes -2

Quality attributes often impart a "language" of things that you would look for.

- **Security involves security levels, authenticated users, audit trails, firewalls, and the like.**
- **Performance brings to mind buffer capacities, deadlines, periods, event rates and distributions, clocks and timers, and so on.**
- **Availability conjures up mean time between failure, failover mechanisms, primary and secondary functionality, critical and noncritical processes, and redundant elements.**

Architecture Documentation and Quality Attributes -3

Architecture documentation often contains a mapping to requirements that shows how requirements are satisfied.

Every quality attribute requirement will have a constituency of stakeholders who want to know that it is going to be satisfied.

- **provide a special place in the documentation's introduction that either provides what the stakeholder is looking for, or tells the stakeholder where in the document to find it (documentation roadmap)**

Documenting Fast-Changing Architectures

Document what is true about all versions of your system.

- **Record invariants as you would for any architecture.**
- **This may make your documented architecture more a description of constraints or guidelines that any compliant version of the system must follow.**

Document the ways the architecture is allowed to change.

- **usually mean adding new components and replacing components with new implementations**
- **in the Views and Beyond approach, the place to do this is called the variability guide**

Documenting Architecture in an Agile Environment

Views and Beyond and Agile philosophies agree – If information isn't needed, don't document.

- **Adopt a template or standard organization to capture your design decisions.**
- **Plan to document a view if (but only if) it has a strongly identified stakeholder constituency.**
- **Fill in the sections of the template only if writing down this information will make it easier (or cheaper or make success more likely) for someone downstream doing their job.**
- **Produce just enough design information to allow you to move on to code.**
- **Don't feel obliged to fill up all sections of the template**
- **When documenting a view, the primary presentation may consist of a digital picture of the whiteboard.**

Architecture Reconstruction

Obtaining the as-built architecture from an existing system

To document an architecture where the documentation never existed or where it has become hopelessly out of date

To ensure conformance between the as-built architecture and the as-designed architecture

Reverse engineer from existing system artifacts

- **(semi)automated extraction tools**
- **probe the original design intent of the architect**

Architectures Are Abstractions

Cannot be seen in the low-level implementation details

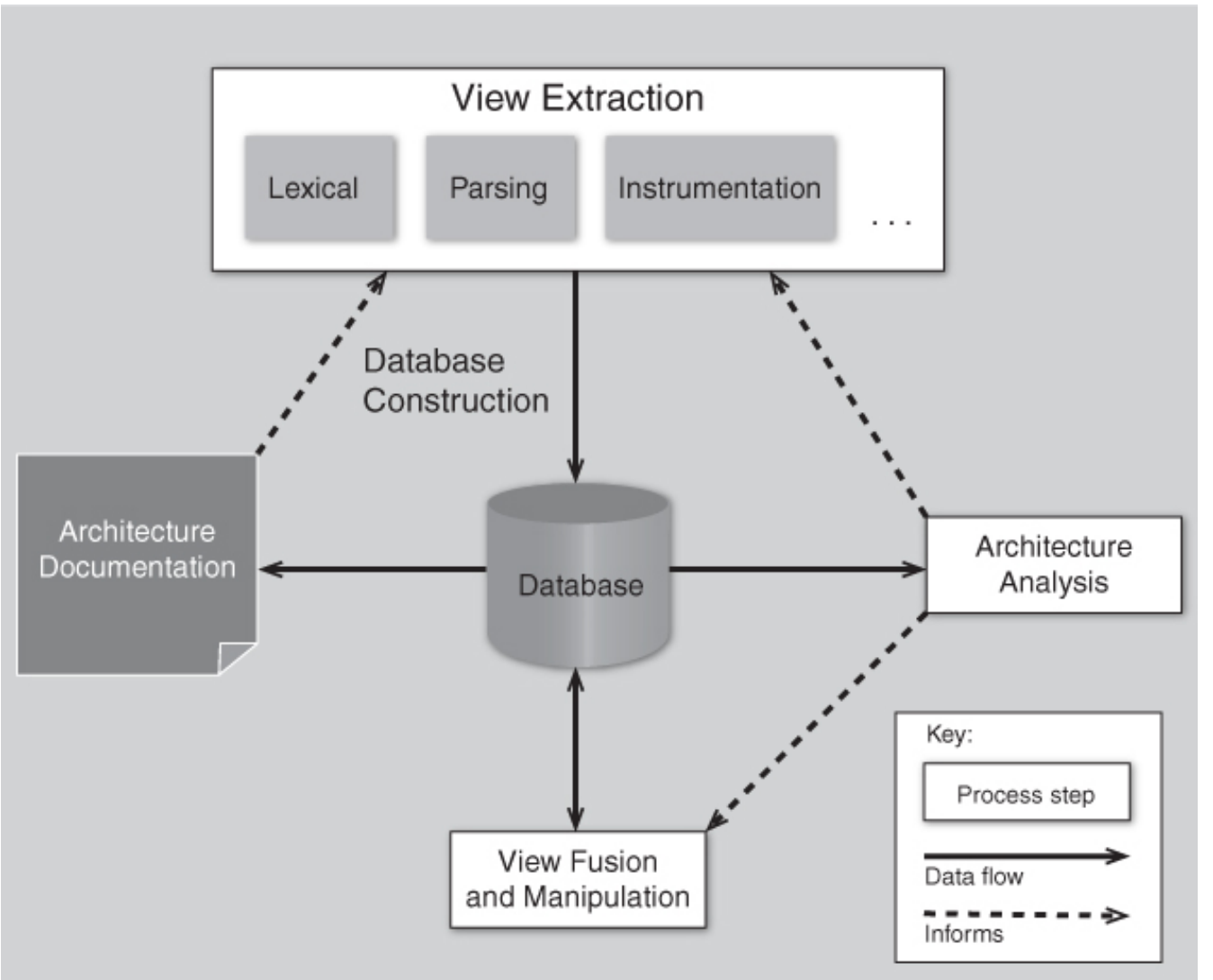
Tools aggregate abstractions

- **not a pancea**
- **no programming language construct for layer or connector or …**

Architecture reconstruction is an interpretive, interactive, iterative process

Workbench – open, integration framework

Architecture Reconstruction Process



Architecture Reconstruction Guidelines

Have a goal and a set of objectives or questions in mind before undertaking an architecture reconstruction project.

Obtain some representation, however coarse, of the system before beginning detailed reconstruction.

- **e.g., identifying layers**

Disregard existing inaccurate documentation.

- **may be useful for generating high-level views**

Involve people familiar with the system early.

Architecture Evaluation

By the designer within the design process

By peers within the design process

By outsiders once the architecture has been designed

ATAM

Architecture Tradeoff Analysis Method

Requires participation and cooperation of

- **evaluation team**
- **project decision makers**
  - **project manager**
  - **customer**
  - **architect**
- **architecture stakeholders**
  - **state specific quality attribute goals the architecture should meet to be considered successful**

ATAM Evaluation Team Roles

Role	Responsibilities
Team Leader	Sets up the evaluation; coordinates with client, making sure client's needs are met; establishes evaluation contract; forms evaluation team; sees that final report is produced and delivered (although the writing may be delegated)
Evaluation Leader	Runs evaluation; facilitates elicitation of scenarios; administers scenario selection/prioritization process; facilitates evaluation of scenarios against architecture; facilitates on-site analysis
Scenario Scribe	Writes scenarios on flipchart or whiteboard during scenario elicitation; captures agreed-on wording of each scenario, halting discussion until exact wording is captured
Proceedings Scribe	Captures proceedings in electronic form on laptop or workstation: raw scenarios, issue(s) that motivate each scenario (often lost in the wording of the scenario itself), and resolution of each scenario when applied to architecture(s); also generates a printed list of adopted scenarios for handout to all participants
Questioner	Raises issues of architectural interest, usually related to the quality attributes in which he or she has expertise

ATAM Outputs -1

Concise presentation of the architecture

- **one-hour presentation**

Articulation of the business goals

Prioritized quality attribute requirements expressed as QA scenarios

A set of risks and non-risks

- **architectural decisions that may lead to undesirable consequences in light of stated QA requirements**
- **safe architectural decisions**

ATAM Outputs -2

A set of risk themes

- **overarching themes that identify system weaknesses**
- **in the architecture**
- **in the architecture process**
- **in the team**

Mapping of architectural decisions to quality requirements

A set of identified sensitivity and tradeoff points

ATAM Phases

Phase	Activity	Participants	Typical Duration
0	Partnership and preparation	Evaluation team leadership and key project decision makers	Proceeds informally as required, perhaps over a few weeks
1	Evaluation	Evaluation team and project decision makers	1–2 days followed by a hiatus of 1–3 weeks
2	Evaluation (continued)	Evaluation team, project decision makers, and stakeholders	2 days
3	Follow-up	Evaluation team and evaluation client	1 week

Software Architecture Topics

Introduction to Architecture

Quality Attributes

- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Architecture Governance

The practice and orientation by which enterprise architectures and other architectures are managed and controlled

- **Open Group**

Implement a system of controls over the creation and monitoring of all architectural components and activities

Implement a system to ensure compliance with internal/external standards and regulatory obligations

Establish processes that support effective management of the above processes within agreed parameters

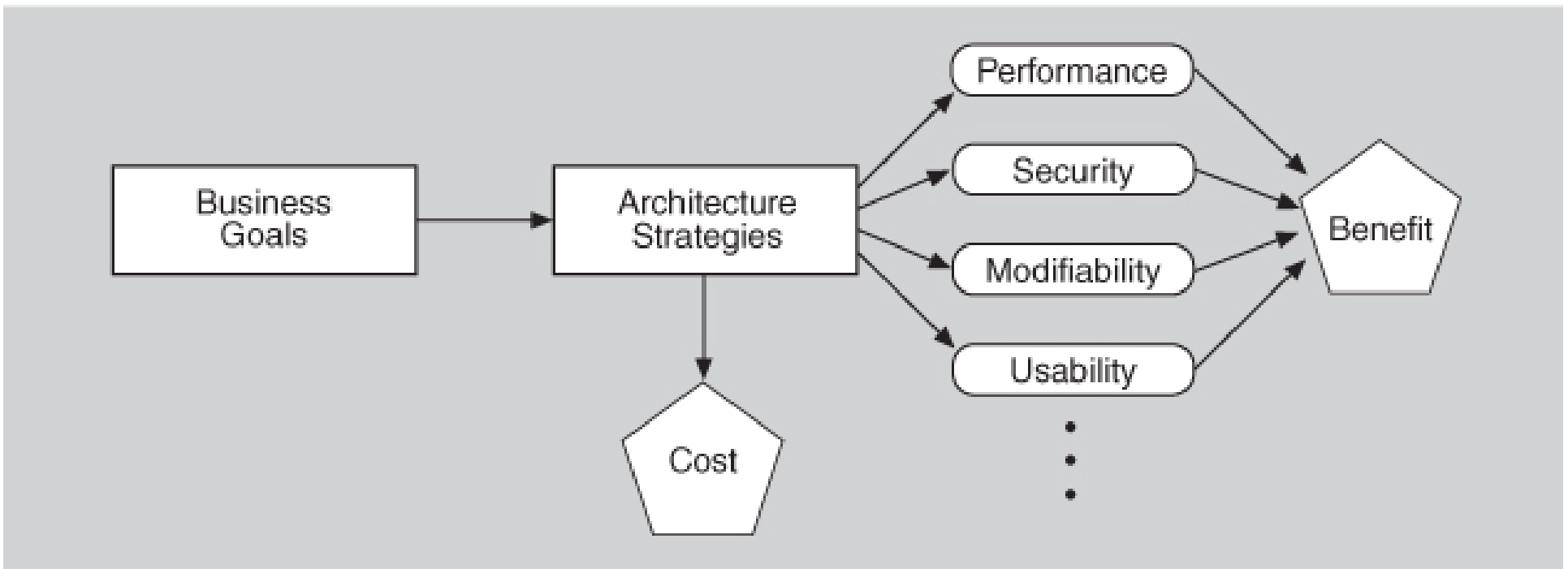
Develop practices that ensure accountabiliy to a clearly identified stakeholder community

Architecture and Business

Perhaps the most important job of an architect is to be a fulcrum where business and technical decisions meet and interact…

What are the economic implications of an architectural decision?

Cost/Benefit and Architecture



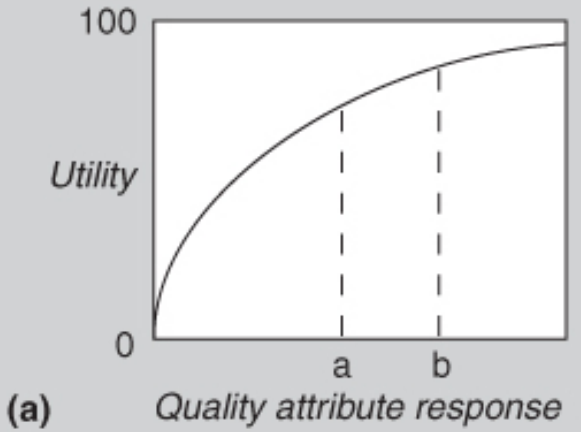
Utility Response Curves

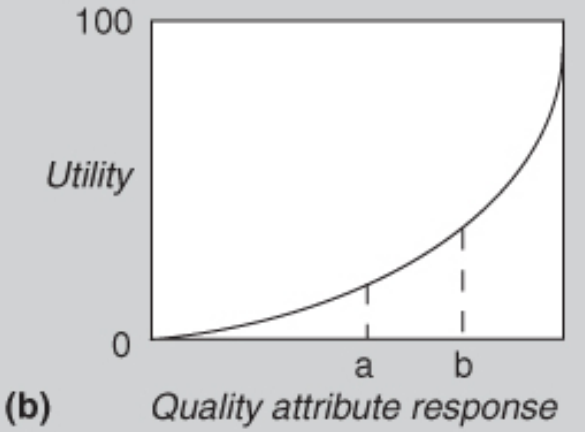
Each scenario's stimulus-response pair provides some utility (value) to stakeholders

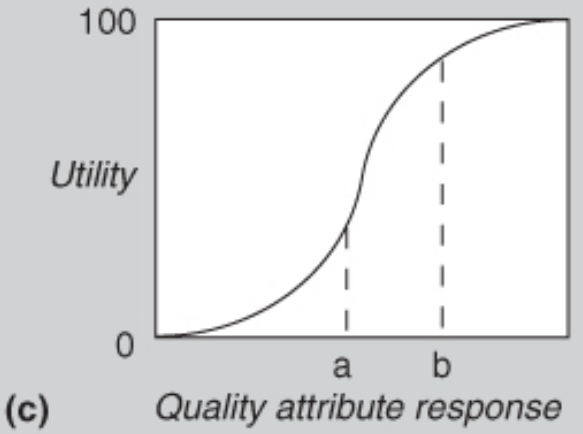
The utility of different possible values for the response can be compared

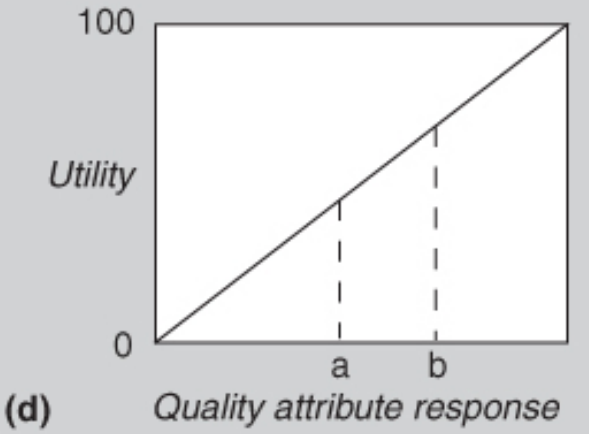
Absolute numbers are not necessary to compare alternatives…

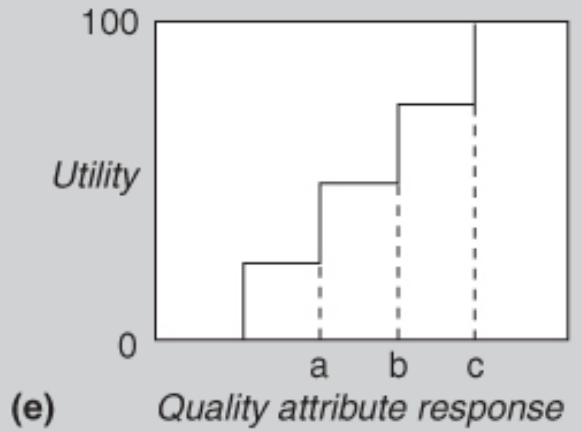
- **human beings are better at comparative estimation**











Some Sample Utility-Response Curves

Determining Benefit

For each architectural strategy i, its benefit B_i of j scenarios (each with weight W_j) is

$$B_i = \Sigma_j (b_{i,j} \times W_j)$$

Each $b_{i,j}$ is calculated as the change in utility brought about by the architectural strategy

$$b_{i,j} = U_{expected} - U_{current}$$

Value for cost is the ratio of the total benefit to the cost of implementing

$$VFC = B_i / C_i$$

Best and Worst Cases

Best-case quality attribute level – that above which the stakeholders foresee no further utility

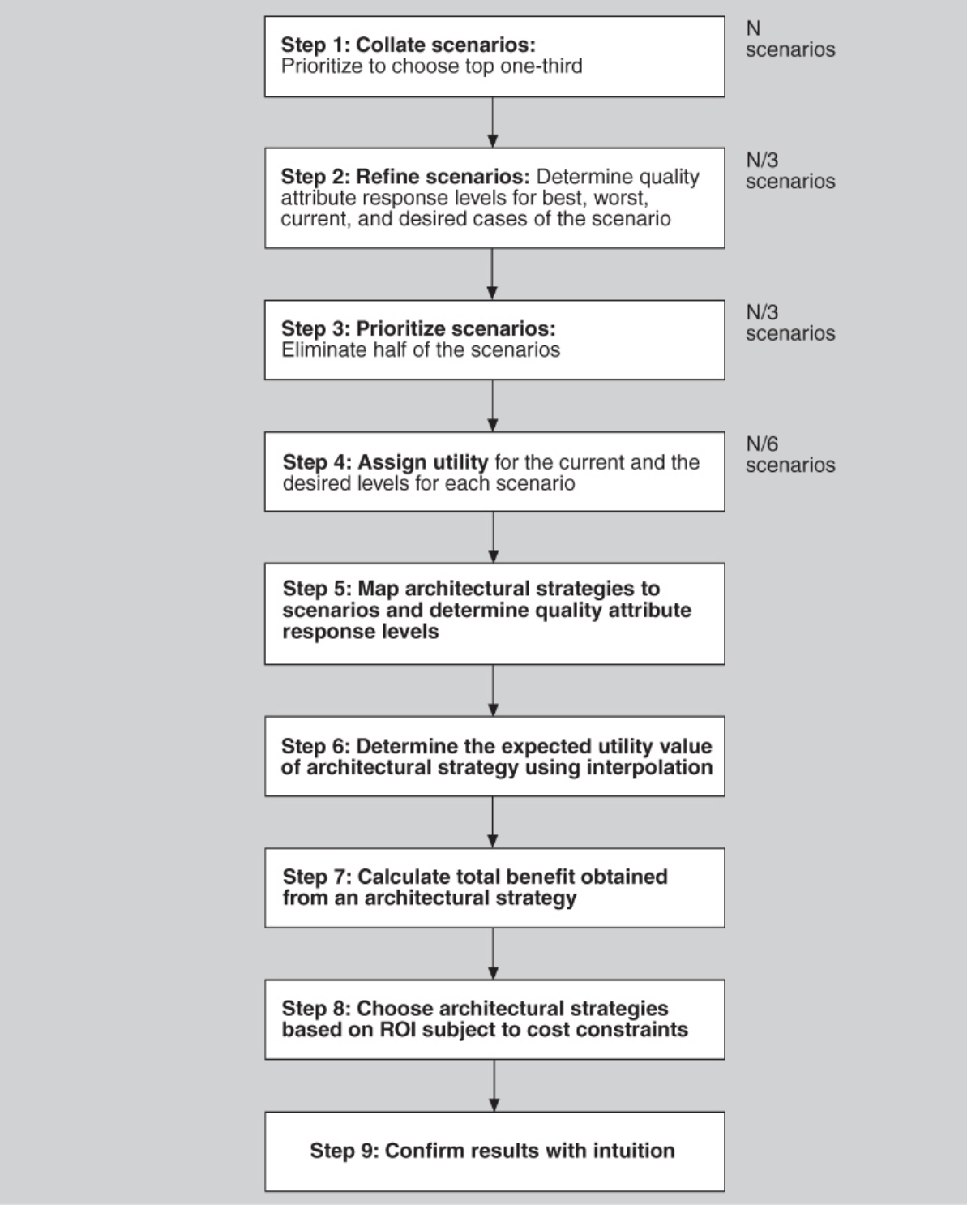
Worst-case quality attribute level – the minimum threshold above which a system must perform, otherwise it is of no value to the stakeholders

Current quality attribute level

Desired quality attribute level

Anchor the utility levels on a scale of 0-100 with the worst and best cases

Cost Benefit Analysis Method (CBAM)



CBAM 1-2

1. Collate scenarios
- **contribute new scenarios**
- **proritize scenarios**
- **choose the top third for further study**

2. Refine scenarios
- **elicit worst, best, current, and desired cases**

Scenario	Worst Case	Current	Desired	Best Case
Scenario #17: Response to user input	12 seconds	1.5 seconds	0.5 seconds	0.1 seconds
…				

CBAM 3-5

3. Prioritize scenarios

- **distribute 100 votes by each stakeholder among scenarios based on desired response**
- **choose half of scenarios for further analysis**

4. Assign utility

Scenario	Worst Case	Current	Desired	Best Case
Scenario #17: Response to user input	12 seconds	1.5 seconds	0.5 seconds	0.1 seconds
	Utility 5	Utility 50	Utility 80	Utility 85

5. Map architectural strategies to scenarios and determine their expected QA response levels.

CBAM 6-7

6. Determine the utility of the expected QA response levels by interpolation.

7. Caculate the total benefit obtained from an architectural strategy.

- **subtract the utility value of the current level from the expected level**
- **normalize it using the votes from step 3**
- **sum the benefit due to a particular architectural strategy across scenarios and QAs**

CBAM 8-9

8. Choose architectural strategies based on VFC subject to cost and schedule constraints.

- **rank-order the architectural strategies according to VFC**
- **choose the top ones until budget or schedule is exhausted**

9. Confirm results with intuition.

- **are the architectural strategies aligned with the organization's business goals?**

Software Architecture Topics

Introduction to Architecture

Quality Attributes

- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Software Product Lines

A set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

- **SEI**

Core assets

- **reusable assets based on a common architecture and the software elements that populate that architecture**
- **includes designs and their documentation, user manuals, project management artifacts, software test plans and test cases, …**

Clone-and-Own

Need a variant of an existing system…

Copy the module (clone it)

Make the necessary changes

The new project owns the new version…

Clone-and-own does not scale.

Potential for Reuse

Requirements

Architectural design

Software elements

Modeling and analysis

Testing

Project planning artifacts

Reuse – Promise Exceeds Payoff

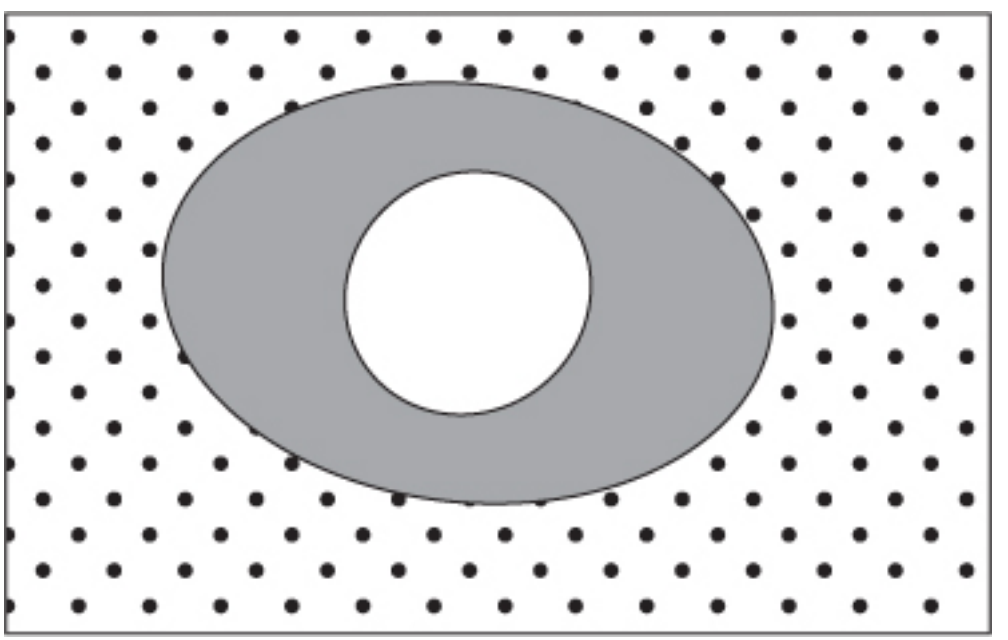
Reuse libraries

- **too sparse – nothing of use to reuse**
- **too rich – hard to understand and search (information retrieval problem)**
- **elements too small – easier to rewrite**
- **elements too large – difficult to understand, adapt**
- **hazy pedigree**
- **written for a different architectural model**

Software product lines make reuse work by establishing a strict context for it. The architecture is defined; the functionality is set; the quality attributes are known. Nothing is placed in the reuse library (core asset base) that was not built to be reused in that product line. Product lines work by relying on strategic, not opportunistic, reuse.

Product Line Scope

The problem in defining scope is not in finding commonality – it's finding commonality that can be exploited to substantially reduce the cost of constructing the systems that an organization intends to build.



Architectural Variation Mechanisms

Software product lines rely on identifying and supporting variation points

- **vary only in small, well-defined ways**

Inclusion or omission of elements

Inclusion of a different number of replicated elements

- **e.g., adding more servers**

Selection of different versions of elements that have the same interface but different behavioral or quality attribute characteristics

- **select at compile time, build time, or runtime**

Common Variation Mechanisms

Variation Mechanism	Properties Relevant to Building the Core Assets	Properties Relevant to Exercising the Variation Mechanism When Building Products
Inheritance; specializing or generalizing a particular class	**Cost:** Medium **Skills:** Object-oriented languages	**Stakeholder:** Product developers **Tools:** Compiler **Cost:** Medium
Component substitution	**Cost:** Medium **Skills:** Interface definitions	**Stakeholder:** Product developer, system administrator **Tools:** Compiler **Cost:** Low
Add-ons, plug-ins	**Cost:** High **Skills:** Framework programming	**Stakeholder:** End user **Tools:** None **Cost:** Low
Templates	**Cost:** Medium **Skills:** Abstractions	**Stakeholder:** Product developer, system administrator **Tools:** None **Cost:** Medium
Parameters (including text preprocessors)	**Cost:** Medium **Skills:** No special skills required	**Stakeholder:** Product developer, system administrator, end user **Tools:** None **Cost:** Low
Generator	**Cost:** High **Skills:** Generative programming	**Stakeholder:** System administrator, end user **Tools:** Generator **Cost:** Low
Aspects	**Cost:** Medium **Skills:** Aspect-oriented programming	**Stakeholder:** Product developer **Tools:** Aspect-oriented language compiler **Cost:** Medium
Runtime conditionals	**Cost:** Medium **Skills:** No special skills required	**Stakeholder:** None **Tools:** None **Cost:** No development cost; some performance cost
Configurator	**Cost:** Medium **Skills:** No special skills required	**Stakeholder:** Product developer **Tools:** Configurator **Cost:** Low to medium

Computing Benefit for Architectural Variation Points

CBAM uses stakeholders to jointly work out utility.

- **subjective, intuitive, imprecise measures**

Product-line architectures have variation points that allow tailoring in pre-planned ways.

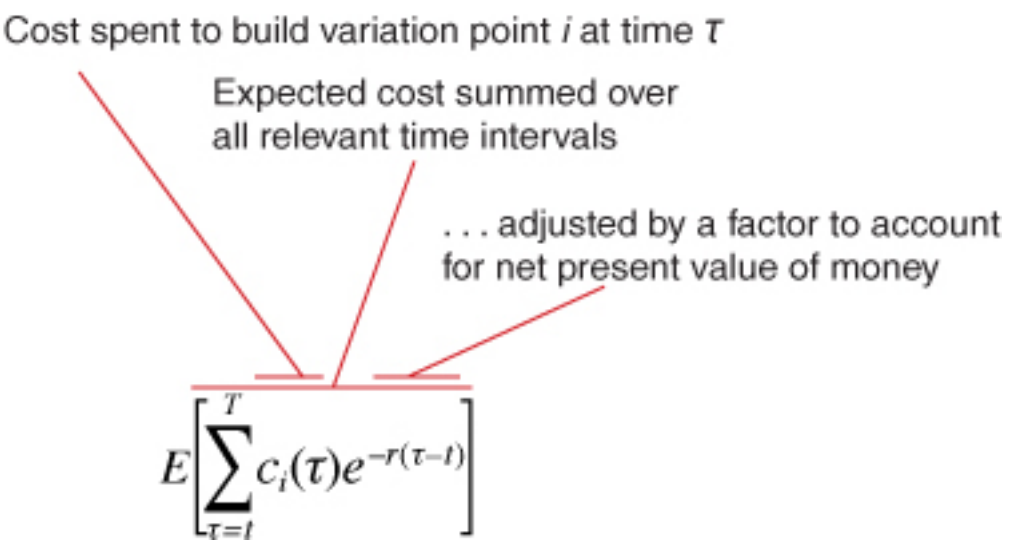
John McGregor's formula for modeling the marginal value of building an additional variation point to the architecture

$$v_i(t,T) = \max(0, -E\left[\sum_{\tau=t}^{T} c_i(\tau)e^{-r(\tau-t)}\right] + P_{i,T}E[\sum_{k}\max(0,\sum_{\tau=T}^{T^*} X_{i,k}(\tau)e^{-r(\tau-t)})])$$

McGregor's First Term

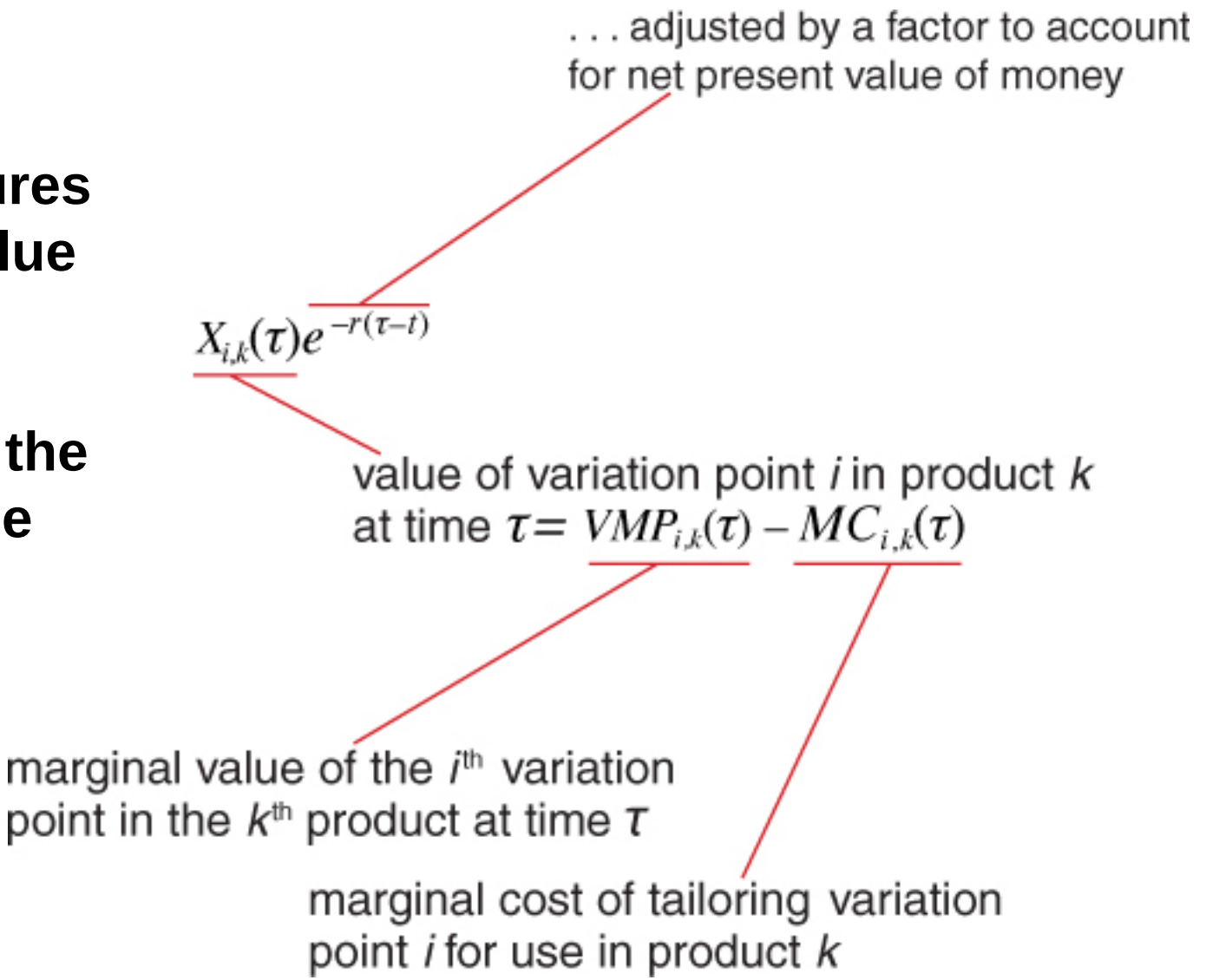
Measures the expected cost of building variation point I over a time period from now until time T

SYMBOLS FOR TIME
τ = time variable
t = time now
T = target date
T^* = modeling limit (t=forever)
i = index over variation points
r = assumed interest rate



McGregor's Second Term -1

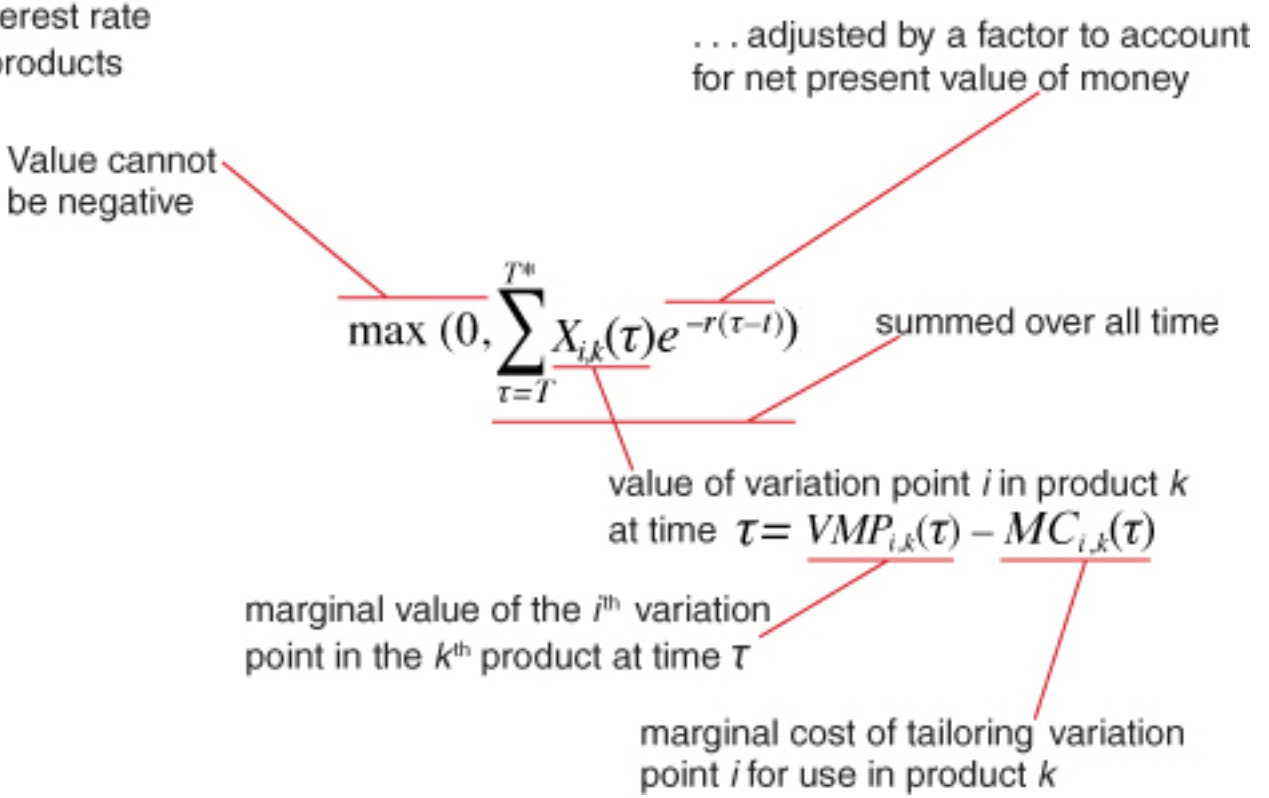
Evaluates the benefit – measures the marginal value of the variation point to the kth product, minus the cost of using the variation point



McGregor's Second Term -2

SYMBOLS FOR TIME
τ = time variable
t = time now
T = target date
T^* = modeling limit (t=forever)
i = index over variation points
r = assumed interest rate
k = index over products

Account for net present value of money



McGregor's Second Term -3

Sum over all products k in the product line and multiply by the probability that the variation point will be ready when needed

$$P_{i,T} E[\sum_{k} \max(0, \sum_{\tau=T}^{T^*} X_{i,k}(\tau) e^{-r(\tau-t)})])$$

probability that variation point *i* will be ready for use by time *T*

value of variation point *i* in product *k* over all time . . .

. . . and over all products

Software Architecture Topics

Introduction to Architecture

Quality Attributes

- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Essential Characteristics of Cloud Computing

On-demand self-service

Ubiquitous network access

Resource pooling

Location independence

Rapid elasticity

Measured service

Multi-tenancy

Cloud Economies of Scale

Cost of power

Infrastructure labor costs

Security and reliability

Hardware costs

Use of equipment

- **random access, time of day, time of year, resource usage patterns, uncertainty**

Multi-tenancy

- **reduction in costs for application update and management**

Cloud Service Models

Software as a Service (SaaS)

- **consumer is an end user**

Platform as a Service (PaaS)

- **integrated stack to develop and deploy applications**
- **consumer is a developer or system admin**
- **consumer controls deployed applications**

Infrastructure as a Service (IaaS)

- **virtualized computation, networking, and file system**
- **consumer is a developer or system admin**
- **consumer deploys and runs arbitrary software**
- **virtual machine with hypervisor**

Performance in the Cloud

The cloud provides an elastic host.

- **additional resources can be acquired as needed**

Application should be aware of current and projected resource usage.

Availability in the Cloud

The cloud is always assumed to be available… but everything can fail.

Netflix example

- **hosted on Amazon EC2 cloud service**
- **99.95% guarantee of service – April 21, 2011, had a four-day sporadic outage**
- **availability tactics used by Netflix**
  - **stateless services**
  - **data stored across zones**
  - **graceful degradation (fail fast, fallbacks, feature removal)**

Edge-Dominant Systems

Depend crucially on the inputs of users for success

Wikipedia, YouTube, Twitter, Facebook, Flickr, …

Web 2.0

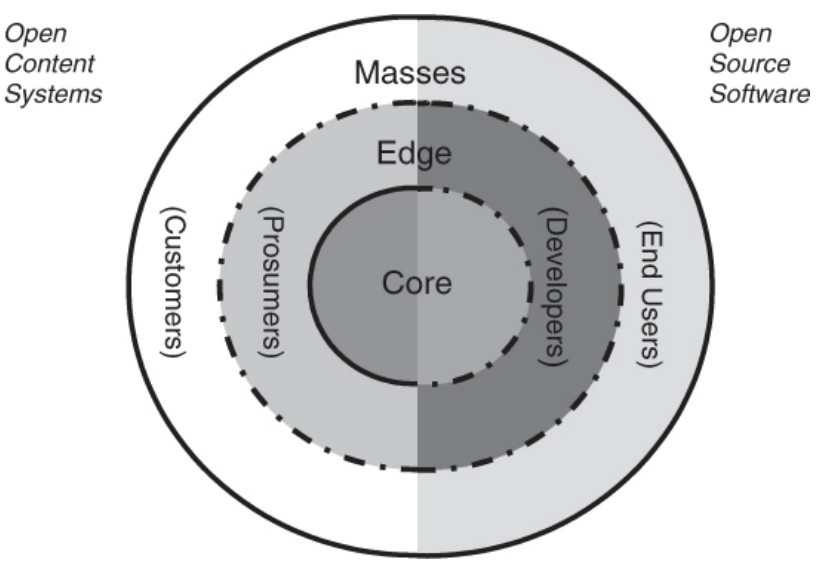
- **The old web was about going to web pages for static information.**
- **The new web is about participating in the information creation and even becoming part of its organization.**

The Ecosystem of Edge-Dominant Systems

All successful edge-dominant systems (and the organizations that develop and use these systems) share a common ecosystem structure.

The Metropolis structure

- **not an architecture diagram**
- **represents three communities of stakeholders**



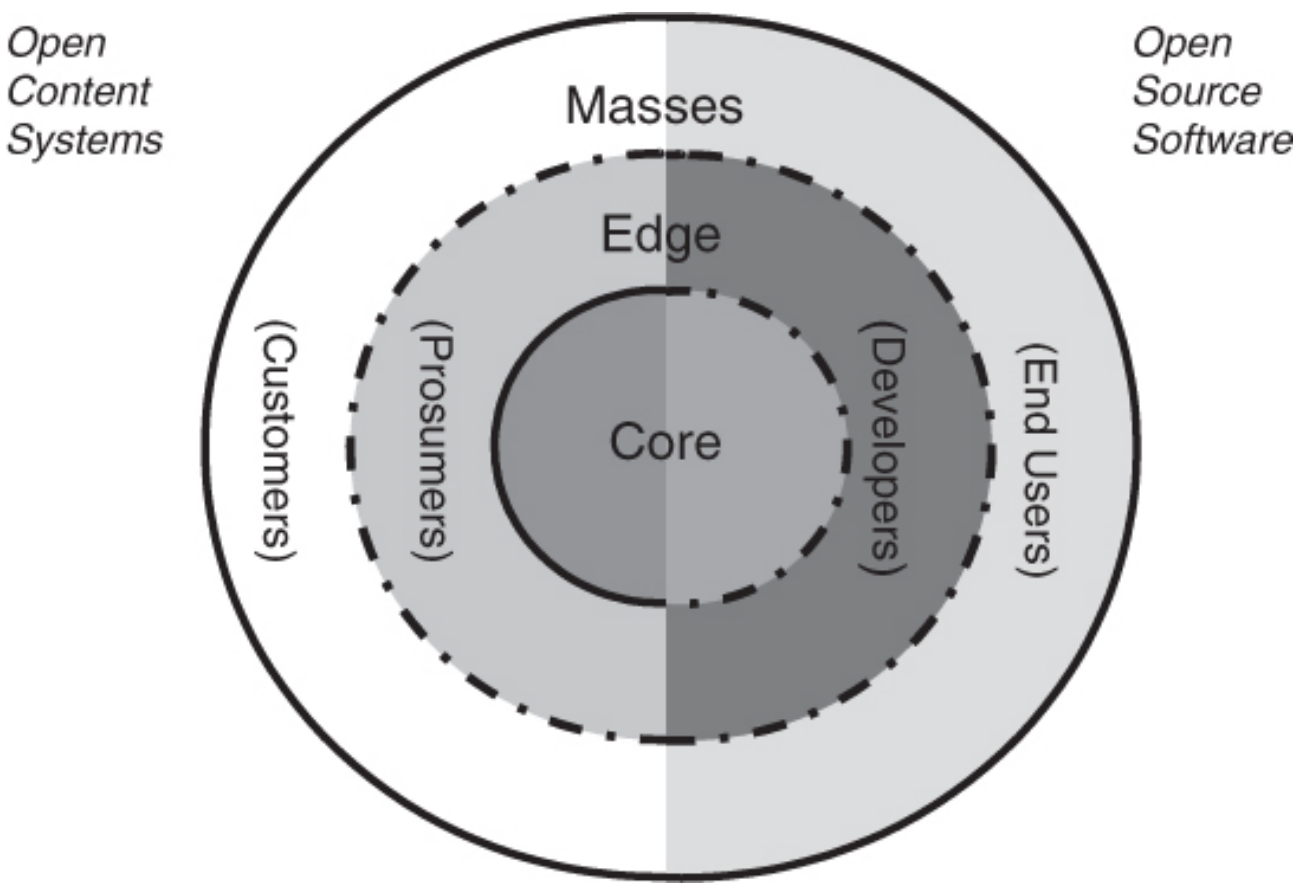
The Metropolis Structure

Customers and end users

Developers

Prosumers

- **consume and produce content**



Architecture Implications of Edge-Dominant Systems

Successful edge-dominant systems bifurcate

- **a core (kernel) infrastructure**
- **a set of peripheral functions or services built on the core**

Core requirements deliver little or no end-user value and focus on quality attributes and tradeoffs.

Periphery requirements are unknowable because they are contributed by the peer network.

Core Implications

The core needs to be highly modular.

- **it provides the foundation for the achievement of quality attributes**

The core must be highly reliable.

The core must be highly robust with respect to errors in its environment.

- **the core will undoubtedly be supporting flawed periphery components**

Questions and Answers