Software Architectures

SECLZG651/SSCLZG653

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Parthasarathy

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

SECLZG651/SSCLZG653 – CS#6

ASRs and Agility

Agenda for CS #6

1) Recap of CS#5
2) Gathering ASRs from requirements documents
3) Gathering ASRs by interviewing Stakeholders
4) Gathering ASRs by understanding the Business Goals
5) Capturing ASRs in utility tree
6) Tying the methods together
7) Architecture and Agile
8) Q&A!

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Architecturally Significant Requirements (ASRs)

Requirements

- Architectures exist to build systems that satisfy requirements.
- But, to an architect, not all requirements are created equal.
- An *architecturally significant requirement* (ASR) is a requirement that will have a *profound effect* on the architecture.
- How do we find those?

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

ASRs and Requirements Documents

- An obvious location to look for candidate ASRs is in the requirements documents or in user stories.
- Requirements should be in requirements documents!
- Unfortunately, this is not usually the case.
- Why?

Don’t Get Your Hopes Up

innovate achieve lead

- Many projects don’t create or maintain the detailed, high-quality requirements documents.
- Standard requirements pay more attention to functionality than quality attributes.
- Most of what is in a requirements specification does not affect the architecture.
- No architect just sits and waits until the requirements are “finished” before starting work. The architect *must* begin while the requirements are still in flux.

Don't Get Your Hopes Up

- Quality attributes, when captured at all, are often captured poorly.
  - "The system shall be modular"
  - "The system shall exhibit high usability"
  - "The system shall meet users' performance expectations"
- 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 itself
  - Developmental qualities (such as teaming) are also out of scope

Sniffing Out ASRs

innovate achieve lead

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)

Gathering ASRs from Stakeholders

- Say your project won't have the QAs nailed down by the time you need to start your design work.
- What do you do?
- Stakeholders often have *no idea* what QAs they want in a system
  - if you insist on quantitative QA requirements, you're likely to get numbers that are arbitrary.
  - at least some of those requirements will be very difficult to satisfy.
- Architects often have very good ideas about what QAs are reasonable to provide.
- Interviewing the relevant stakeholders is the surest way to learn what they know and need.

Gathering ASRs from Stakeholders

- The results of stakeholder interviews should include
  - a list of architectural drivers
  - a set of QA scenarios that the stakeholders (as a group) prioritized.
- This information can be used to:
  - refine system and software requirements
  - understand and clarify the system's architectural drivers
  - provide rationale for why the architect subsequently made certain design decisions
  - guide the development of prototypes and simulations
  - influence the order in which the architecture is developed.

Quality Attribute Workshop (QAW)

innovate achieve lead

- The QAW is a facilitated, stakeholder-focused method to generate, prioritize, and refine quality attribute scenarios before the software architecture is completed.
- The QAW is focused on system-level concerns and specifically the role that software will play in the system.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

QAW Steps

innovate achieve lead

- **Step 1: QAW Presentation and Introductions.**
  - QAW facilitators describe the motivation for the QAW and explain each step of the method.
- **Step 2: Business/Mission Presentation.**
  - The stakeholder representing the business concerns behind the system presents the system's business context, broad functional requirements, constraints, and known quality attribute requirements.
  - The quality attributes that will be refined in later steps will be derived largely from the business/mission needs presented in this step.
- **Step 3: Architectural Plan Presentation.**
  - The architect will present the system architectural plans as they stand.
  - This lets stakeholders know the current architectural thinking, to the extent that it exists.
- **Step 4: Identification of Architectural Drivers.**
  - The facilitators will share their list of key architectural drivers that they assembled during Steps 2 and 3, and ask the stakeholders for clarifications, additions, deletions, and corrections.
  - The idea is to reach a consensus on a distilled list of architectural drivers that includes overall requirements, business drivers, constraints, and quality attributes.

QAW Steps

innovate achieve lead

- **Step 5: Scenario Brainstorming.**
  - Each stakeholder expresses a scenario representing his or her concerns with respect to the system.
  - Facilitators ensure that each scenario has an explicit stimulus and response.
  - The facilitators ensure that at least one representative scenario exists for each architectural driver listed in Step 4.
- **Step 6: Scenario Consolidation.**
  - Similar scenarios are consolidated where reasonable.
  - Consolidation helps to prevent votes from being spread across several scenarios that are expressing the same concern.
- **Step 7: Scenario Prioritization.**
  - Prioritization of the scenarios is accomplished by allocating each stakeholder a number of votes equal to 30 percent of the total number of scenarios
- **Step 8: Scenario Refinement.**
  - The top scenarios are refined and elaborated.
  - Facilitators help the stakeholders put the scenarios in the six-part scenario form of source-stimulus-artifact-environment-response-response measure.

ASRs from Business Goals

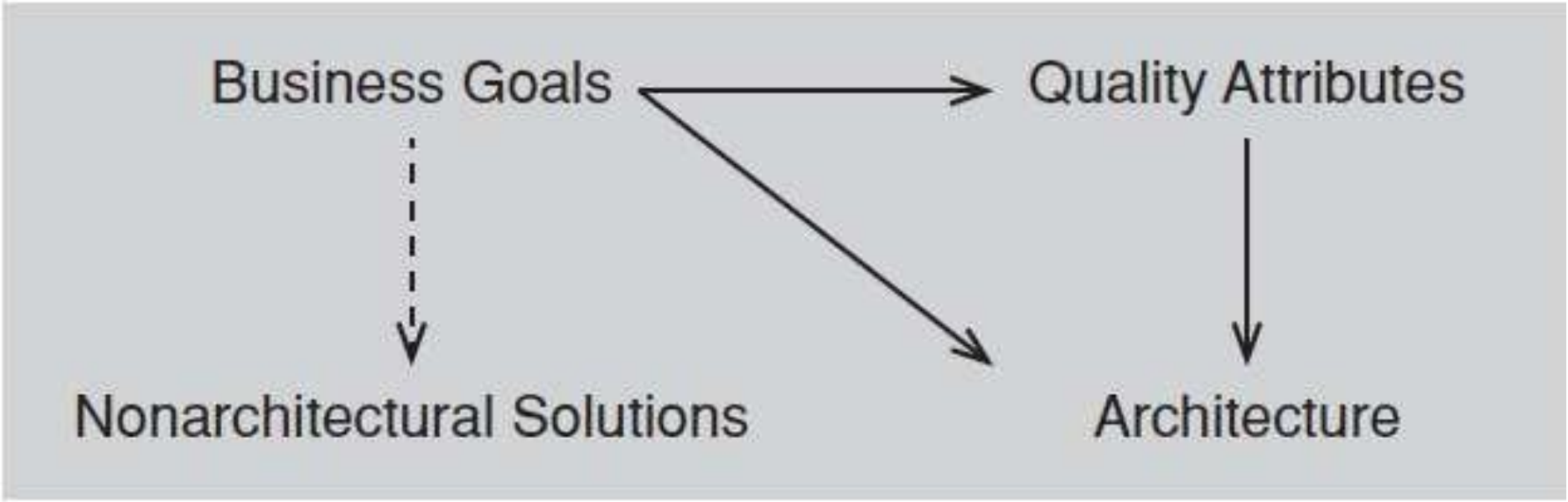


FIGURE 3.2 Some business goals may lead to quality attribute requirements (which lead to architectures), or lead directly to architectural decisions, or lead to nonarchitectural solutions.

Categories of Business Goals, to Aid in Elicitation

innovate achieve lead

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

Expressing Business Goals

Business goal scenario, 7 parts

- *Goal-source*
  - The people or written artifacts providing the goal.
- *Goal-subject*
  - The stakeholders who own the goal and wish it to be true.
  - Each stakeholder might be an individual or the organization itself
- *Goal-object*
  - The entities to which the goal applies.
- *Environment*
  - The context for this goal
  - Environment may be social, legal, competitive, customer, and technological

Expressing Business Goals

Business goal scenario, 7 parts

- *Goal*
  - Any business goal articulated by the goal-source.
- *Goal-measure*
  - A testable measurement to determine how one would know if the goal has been achieved. The goal-measure should usually include a time component, stating the time by which the goal should be achieved.
- *Pedigree and value*
  - The degree of confidence the person who stated the goal has in it
  - The goal's volatility and value.

PALM: A Method for Eliciting Business Goals

- PALM (Pedigreed Attribute eLicitation Method) is a seven-step method.
- Nominally carried out over a day and a half in a workshop.
- Attended by architect(s) and stakeholders who can speak to the relevant business goals.

PALM Steps

- *PALM overview presentation*
  - Overview of PALM, the problem it solves, its steps, and its expected outcomes.
- *Business drivers presentation.*
  - Briefing of business drivers by project management
  - What are the goals of the customer organization for this system?
  - What are the goals of the development organization?
- *Architecture drivers presentation*
  - Briefing by the architect on the driving business and quality attribute requirements: the ASRs.
- *Business goals elicitation*
  - Business goals are elaborated and expressed as scenarios.
  - Consolidate almost-alike business goals to eliminate duplication.
  - Participants prioritize the resulting set to identify the most important goals.

PALM Steps

innovate achieve lead

- *Identification of potential quality attributes from business goals.*
  - For each important business goal scenario, participants describe a quality attribute that (if architected into the system) would help achieve it. If the QA is not already a requirement, this is recorded as a finding.
- *Assignment of pedigree to existing quality attribute drivers.*
  - For each architectural driver identify which business goals it is there to support.
  - If none, that's recorded as a finding.
  - Otherwise, we establish its pedigree by asking for the source of the quantitative part.
- *Exercise conclusion*
  - Review of results, next steps, and participant feedback.

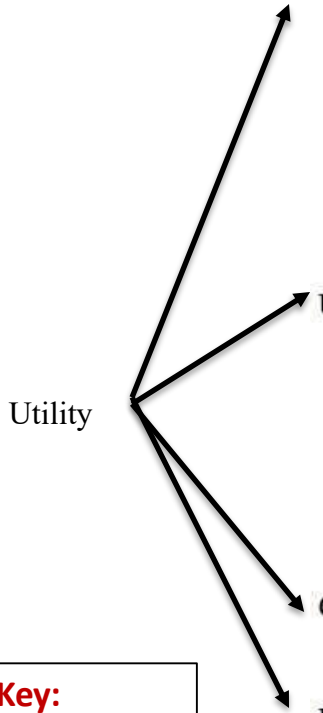
Capturing ASRs in a Utility Tree

- An ASR must have the following characteristics:
- *A profound impact on the architecture*
- Including this requirement will very likely result in a different architecture than if it were not included.
- *A high business or mission value*
  - If the architecture is going to satisfy this requirement it must be of high value to important stakeholders.

Utility Tree

- A way to record ASRs all in one place.
- Establishes priority of each ASR in terms of
  - Impact on architecture
  - Business or mission value
- ASRs are captured as scenarios.
- Root of tree is placeholder node called “Utility”.
- Second level of tree contains broad QA categories.
- Third level of tree refines those categories.

Utility Tree Example (Sample)



Quality Attribute	Attribute Refinement	ASR
Performance	Transaction response time	A user updates a patient's account in response to a change-of-address notification while the system is under peak load, and the transaction completes in less than 0.75 second. (H,M)
		A user updates a patient's account in response to a change-of-address notification while the system is under double the peak load, and the transaction completes in less than 4 seconds. (L,M)
	Throughput	At peak load, the system is able to complete 150 normalized transactions per second. (M,M)
Usability	Proficiency training	A new hire with two or more years' experience in the business becomes proficient in Nightingale's core functions in less than 1 week. (M,L)
		A user in a particular context asks for help, and the system provides help for that context, within 3 seconds. (H,M)
	Normal operations	A hospital payment officer initiates a payment plan for a patient while interacting with that patient and completes the process without the system introducing delays. (M,M)
Configurability	User-defined changes	A hospital increases the fee for a particular service. The configuration team makes the change in 1 working day; no source code needs to change. (H,L)
Maintainability	Routine changes	A maintainer encounters search- and response-time deficiencies, fixes the bug, and distributes the bug fix with no more than 3 person-days of effort. (H,M)
		A reporting requirement requires a change to the report-generating metadata. Change is made in 4 person-hours of effort. (M,L)
	Upgrades to commercial	The database vendor releases a new version that must be

Key:
H=high
M=medium
L=low

Utility Tree: Next Steps

innovate achieve lead

- A QA or QA refinement without any ASR is not necessarily an error or omission
  - Attention should be paid to searching for unrecorded ASRs in that area.
- ASRs that rate a (H,H) rating are the ones that deserve the most attention
  - A very large number of these might be a cause for concern: Is the system is achievable?
- Stakeholders can review the utility tree to make sure their concerns are addressed.

Tying the Methods Together

innovate achieve lead

- How should you employ requirements documents, stakeholder interviews, Quality Attribute Workshops, PALM, and utility trees in concert with each other?
  - If you have a requirements process that gathers, identifies, and prioritizes ASRs, then use that and consider yourself lucky.
  - Otherwise use one or more of the other approaches.
  - If nobody has captured the business goals behind the system you're building, use PALM.
  - If important stakeholders have been overlooked in the requirements-gathering process, use interviews or a QAW.
  - Build a utility tree to capture ASRs along with their prioritization.
  - You can blend all the methods together
    - Use PALM as a "subroutine call" from a QAW for the business goal step
    - Use a quality attribute utility tree as a repository for the scenarios produced by a QAW.
- Pick the approach that fills in the biggest gap in your existing requirements.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Architectures in Agile Projects

Agility

- Agile processes were a response to a need for projects to
  - be more responsive to their stakeholders
  - be quicker to develop functionality that users care about
  - show more and earlier progress in a project's life cycle
  - be less burdened by documenting aspects of a project that would inevitably change.
- These needs are not in conflict with architecture.
- The question for a software project is not "Should I do Agile or
- architecture?" Instead ask:
  - "How much architecture should I do up front versus how much should I defer until the project's requirements have solidified somewhat?"
  - "How much of the architecture should I formally document, and when?"
- Agile and architecture are happy companions for many software
- projects.

Agile Manifesto

innovate achieve lead

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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. [agilemanifesto.org]

Agile

- These processes were initially employed on small- to medium-sized projects with short time frames and enjoyed considerable success.
- They were not often used for larger projects, particularly those with distributed development.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

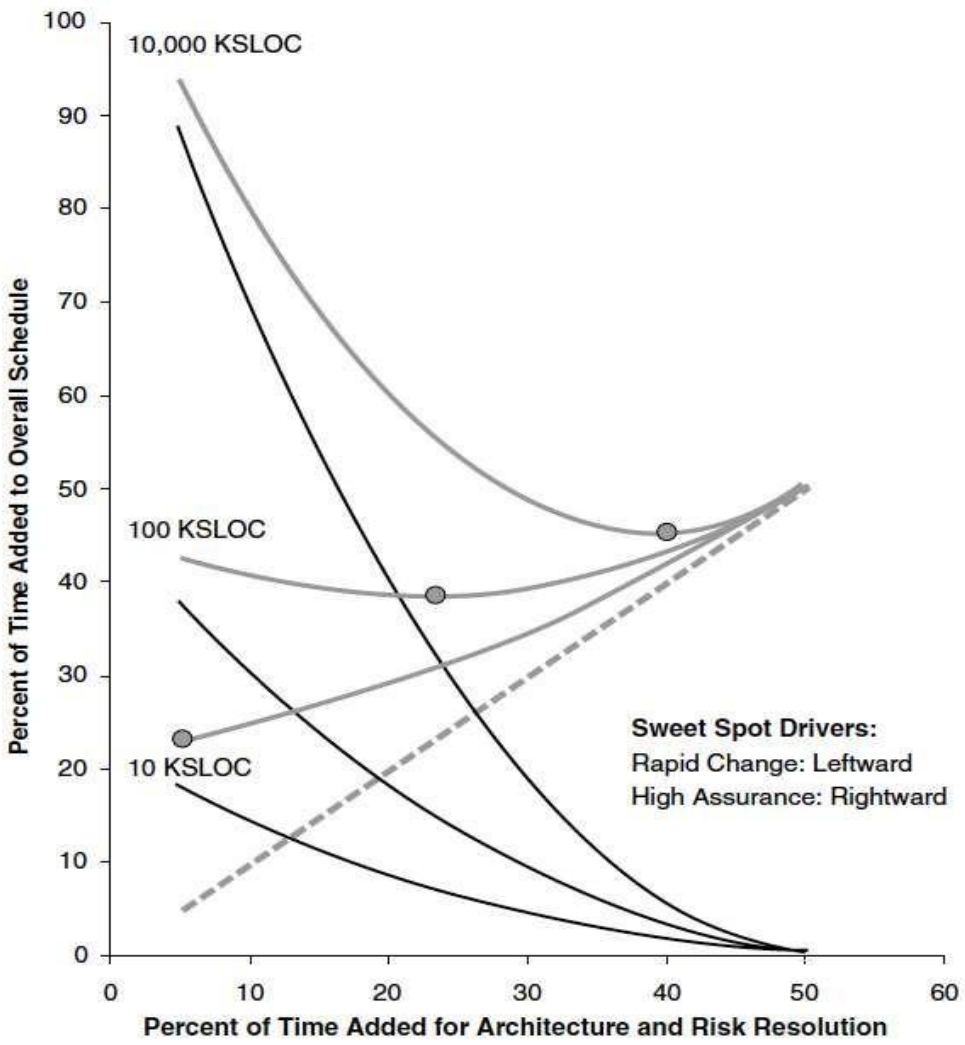
Twelve Agile Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

How Much Architecture?

- There are two activities that can add time to the project schedule:
  - Up-front design work on the architecture and up-front risk identification, planning, and resolution work
  - Rework due to fixing defects and addressing modification requests.
- Intuitively, these two trade off against each other.
- Boehm and Turner plotted these two values against each other for three hypothetical projects:
  - One project of 10 KSLOC
  - One project of 100 KSLOC
  - One project of 1,000 KSLOC

How Much Architecture?



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

How Much Architecture?

innovate achieve lead

- These lines show that there is a sweet spot for each project.
  - For the 10KSLOC project, the sweet spot is at the far left. Devoting much time to up-front work is a waste for a small project.
  - For the 100 KSLOC project, the sweet spot is around 20 percent of the project schedule.
  - For the 1,000 KSLOC project, the sweet spot is around 40 percent of the project schedule.
- A project with a million lines of code is enormously complex.
- It is difficult to imagine how Agile principles alone can cope with this complexity if there is no architecture to guide and organize the effort.

Agility and Architecture Practices

Requirements:

- User stories can be augmented with quality attribute information
- The Initial Requirements Envisioning phase can employ a QAW, or techniques from a QAW.

Architecture Design:

- "Architecture Envisioning" can employ techniques from ADD—perhaps just the first iteration or two.
- As the project matures further iterations can be fleshed out.

Architecture Documentation:

- Write for the reader!
- If the reader doesn't need it, don't write it.
  - But remember that the reader may be a maintainer or other newcomer not yet on the project!

Agility and Architecture Practices

Architecture Evaluation:

- Does architecture evaluation work as part of an Agile process? Absolutely.
- Meeting stakeholders' important concerns is a cornerstone of Agile
- philosophy.
- Our approach to architecture evaluation is exemplified by the Architecture Tradeoff Analysis Method (ATAM).
  - ATAM does not endeavor to analyze all, or even most, of an architecture.
  - The focus is determined by a set of quality attribute scenarios that represent the most important of the concerns of the stakeholders.
- It is easy to tailor a lightweight architecture evaluation.
- Such evaluations can provide valuable input to refactoring and re-
- design.

An Example of Agile Architecting

- WebArrow Web-conferencing system
- To meet stakeholder needs, architect and developers found that they needed to think and work in two different modes at the same time:
  - *Top-down*—designing and analyzing architectural structures to meet the demanding quality attribute requirements and tradeoffs
  - *Bottom-up*—analyzing a wide array of implementation-specific and environment-specific constraints and fashioning solutions to them

Experiments to Make Tradeoffs

innovate achieve lead

- To analyze architectural tradeoffs, the team adopted an agile architecture discipline combined with a rigorous program of experiments.
  - These experiments are called "spikes" in Agile terminology.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Experiments to Make Tradeoffs

- The experiments (spikes) answered questions such as:
  - Would moving to a distributed database from local flat files negatively impact feedback time (latency) for users?
  - What (if any) scalability improvement would result from using mod_perl versus standard Perl?
  - How difficult would the development and quality assurance effort be to convert to mod_perl?
  - How many participants could be hosted by a single meeting server?
  - What was the correct ratio between database servers and meeting servers?

Lesson

- Making architecture processes agile does not require radical re-invention of either Agile practices or architecture methods.
- The WebArrow team's emphasis on experimentation proved the key factor.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

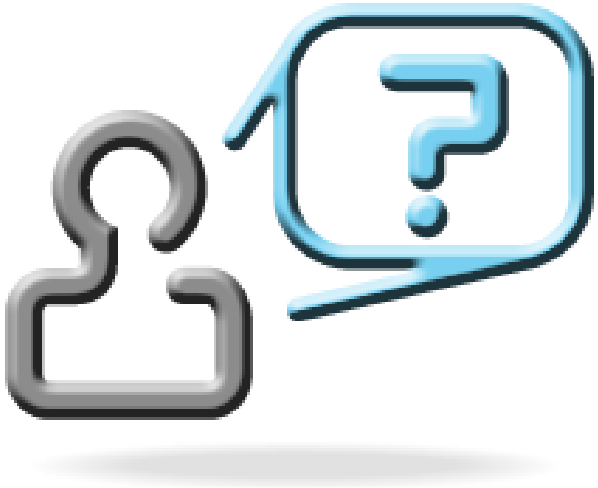
Guidelines for the Agile Architect

- Barry Boehm and colleagues have developed the Incremental Commitment Model to blend agility and architecture.
- This model is based upon the following principles:
  - Commitment and accountability of success-critical stakeholders
  - Stakeholder "satisficing" (meeting an acceptability threshold) based on success-based negotiations and tradeoffs
  - Incremental and evolutionary growth of system definition and stakeholder commitment
  - Iterative system development and definition
  - Interleaved system definition and development allowing early fielding of core capabilities, continual adaptation to change, and timely growth of complex systems without waiting for every requirement and subsystem to be defined
  - Risk management—risk-driven anchor point milestones, which are key to synchronizing and stabilizing all of this concurrent activity

Advice

- If you are building a large, complex system with relatively stable and well-understood requirements and/or distributed development, doing a large amount of architecture work up front will pay off.
- On larger projects with *unstable* requirements, start by quickly designing a candidate architecture even if it leaves out many details.
  - Be prepared to change and elaborate this architecture as circumstances dictate, as you perform your spikes and experiments, and as functional and quality attribute requirements emerge and solidify.
- On smaller projects with uncertain requirements, at least try to get agreement on the major patterns to be employed. Don't spend too much time on architecture design, documentation, or analysis up front.

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956



Thank You for your time & attention !

Contact : parthasarathypd@wilp.bits-pilani.ac.in