Continuous Architecture

© Pierre Pureur & Murat Erder

The opinions and views expressed in this presentation are those of the authors and do not necessarily reflect the position of their employers.
Outline

• **Context** – Brief History and Why

• Continuous Architecture **Principles**
  1. Architect **Products** – not solutions for Projects
  2. Focus on **Quality Attributes** – not on Functional Requirements
  3. Delay Design Decisions Until They Are Absolutely Necessary To Keep The Architecture Manageable
  4. Leverage “The Power Of Small” To Architect For Change
  5. Architect for **Build, Test and Deploy** To Deliver Capabilities Continuously
  6. Model The **Organization Of Your Teams** After The Design Of The System To Promote Interoperability

• Brief **Case Study**

• Some Thoughts on **Modernizing Monolithic** Systems

• Conclusion: **Why does Continuous Architecture Work?**
Context - A Few Thoughts

- Why do we even architect – why not just get on with some *quick design and a few sprints*?
- What is *architecture*?
  - A bag full of tools and the knowledge on when to use them
  - The ability to abstract at the correct level to make and communicate the right decision
  - The ability to plan ahead beyond the current project, phase, iteration
- The basics tenets of good architecture have not changed. What has evolved is the *technology landscape*
  - Increased ability to work at larger scale and in a distributed manner
  - Desire for quicker delivery timelines

What has not changed is the *organizational hurdles* that are the same as in Greek Antiquity
Overview: What is Continuous Architecture?

- The current trend in the industry is away from traditional Enterprise Architecture. We do not believe that the pendulum will swing back to traditional EA, and there is a need for an architectural approach that can encompass Continuous Delivery, providing it with a broader architectural perspective.

- We call this approach “Continuous Architecture”. This is about using the appropriate tools to make the right decisions and support Continuous Delivery, Continuous Integration and Continuous Testing.

- “Continuous Architecture” is an approach based on a toolbox – not a formal process!
Overview

Continuous Architecture (CA) Goals

• To create an architecture that can evolve with applications, that is testable, that can respond to feedback and in fact is driven by feedback

• To make Enterprise Architecture real

• To make solution architecture sustainable

• To create real world, actionable, useful strategies
Continuous Architecture **Principles**

1. Architect **Products** – not solutions for Projects

2. Focus on **Quality Attributes** – not on Functional Requirements

3. Delay **Design Decisions** until They are Absolutely Necessary

4. Architect for **Change** – Leverage “The Power of Small”

5. **Architect for Build, Test and Deploy**

6. Model the **Organization** of your Teams after the Design of the System You are Working on

---

The difference between bottom left and right is how the data is represented in the algorithm, either as a complex number or a real number which reflects the intensity.
CA Principle 1: *Architect Products – Not Just Solutions For Projects*

- It is all about *focus*: Solution Architecture vs. Enterprise Architecture

- Projects rarely exist in isolation – they are often part of a “*Software Product*”

- What is a *Software Product*?
  - This concept forces IT shops to think differently about how they are delivering software
  - Think about your apps like a commercial software company

- What is *Product Management*?
  - Focus on stakeholders, not on plans or budgets
What Exactly is Product Management?

• The concept of product management has been in the industry for decades and focuses on the success of introducing and maintaining a successful commercial product for an organization.

  Product Management is the activity of ‘product’ ownership from Product Conception to Product Withdrawal. It is without any doubt an essential business discipline whereby the Product Manager is analogous to the conductor of an orchestra - without the conductor, uncontrolled pandemonium sets in with each instrument fighting to be heard in continuous cycle of disarray’ – source Productmanager.co.uk

• What does this mean for IT?
  – Product Management as a First Class Citizen
    • Very few IT organizations value the role of a product manager. The first step is to clearly articulate this role and give it teeth. The product manager is peer of the IT owner.
  – Govern properly – Fill or Kill
    • Basically, we are talking about a governance model where all products are reviewed periodically and it is determined either to provide them with additional/continued budget (Fill) or to stop the investment and wind down (Kill).
  – Get the commercials right
    • The commercial aspect for IT is not around producing profit but ability to justify its cost basis.
Case Study: *The “Mobile Shopping” System*

• We will use a simple Case Study to discuss the Continuous Architecture Principles

• The IT group in a large Corporation has received a request to build a *new mobile system* to allow prospective customers to do price comparisons, and to purchase products

• We will follow them through their journey, as they architect, design, build, test and deliver this new “MobileShopping” system using the Continuous Architecture approach

© Pierre Pureur & Murat Erder
Case Study: *Principle 1 - Architect Products – Not Just Solutions For Projects*

How would you get started?

Start Small - *Refactor Existing Architectures Rather Than Creating from Scratch*

Look for existing systems that provide *similar capabilities*
Case Study: **Principle 1 - Architect Products – Not Just Solutions For Projects**

- The team starts small – They want to **refactor existing architectures rather than creating from scratch**
- They discover that the “MobileShopping” system is **very similar to an existing web based system** that allows prospective customers to obtain an on-line quote for the Company’s products.
- They find out that the product manager was planning to implement a **mobile User Interface capability** 6 months from now – so it makes sense to consolidate the two projects.
- Next step is to **consolidate requirements** (both functional and non-functional) between projects, and design an architecture that supports both projects.
CA Principle 2: *Focus on Quality Attributes – not on Functional Requirements*

- Requirements fall into 2 categories:
  - Functional Requirements
  - Non Functional Requirements a.k.a *Quality Attribute Requirements (QAR’s)*

- QAR’s are often *poorly documented*:
  - The system must operate 24/7
  - The system must be extremely user friendly
  - The system must be very fast

- Yet they drive the *architecture design*!
  - Functional Requirements define what the system must do
  - QAR’s define how it does it
  - Clarifying QAR’s is important (see Philippe Kruchten’s story)

- Designing for QAR’s limits the number of *candidate architectures* – usually down to a low number
Case Study: *Principle 2 - Focus on Quality Attributes – not on Functional Requirements*

How would you *focus on Quality Attributes*?

Leverage the *Architecture Trade-off Analysis Method (ATAM) Utility Tree* to better understand Quality Attributes.

Assume that *Performance* is the team’s main area of concern. What would you do?
The team quickly creates a prototype of the “MobileShopping” application. The prototype has very limited functionality - a very small subset of the fields is displayed on a single screen, and there is only one field available for user input.

The team is able to create this prototype in a week, and perform some preliminary performance tests. The results of those tests are encouraging, and the team decides to proceed with detailed design.
CA Principle 3: **Delay Design Decisions Until They Are Absolutely Necessary To Keep The Architecture Manageable**

- Make design decisions only *when facts are known*

- Functional requirements are often *poorly stated*
  - Stakeholders often can’t describe what they want until they see it
  - Interviews often involve proxies rather than the actual stakeholder
  - The Minimum Viable Product strategy can be very effective

- *QAR’s* may also change
  - Beware of *Modifiability!*

- Think *Minimum Viable Architecture (MVA)*
  - Avoid the Big Architecture Up Front (BArF) syndrome!
Case Study: **Principle 3 - Delay Design Decisions Until They Are Absolutely Necessary To Keep The Architecture Manageable**

- Should *caching capabilities* be introduced– “just in case”?

- Should a *configuration engine* (rules engine) be implemented for “*Configurability*”?

**Tip:** *Keep things as simple as possible*. Make a small number of design decisions based on the few facts known at the beginning of the project and avoid design guesses.
Case Study: **Principle 3 - Delay Design Decisions Until They Are Absolutely Necessary To Keep The Architecture Manageable**

- Caching capabilities would add much *complexity to the design* and make *testing and deployment harder*. Also the results of the tests performed using a prototype, are satisfactory and they *delay implementation of caching until it becomes absolutely necessary*.

- The team decides against using a *rules engine* due to concerns about *increased coupling* caused by that component.
CA Principle 4: *Leverage “The Power Of Small” To Architect For Change*

- Change is *unavoidable*
  - How can we create architectures which are resilient to change?

- Base the architecture on *smaller, loosely coupled* components
  - The goal is to replace not modify as new requirements emerge
  - Beware of tight coupling in unexpected places
    - Databases
    - Rules Engines

- Leverage the *Robustness* Principle (Postel’s law)
  - “Be conservative in what you do, be liberal in what you accept from others”
  - Often reworded as “Be conservative in what you send, be liberal in what you accept”

- Design with *Microservices*
  - See next slide
Some Thoughts on Microservices

• Microservices are not “just a REST API”.

• Microservices are not about code base size
  – Think “2 pizzas team” – including everyone (Developers, DBA’s, Operations, etc…)
  – Pain (especially communication pain) is a good indicator

• They increase velocity – and complexity!

• They are about:
  – Functional decomposition
  – Domain driven design
  – Robustness
  – Graceful degradation

• Do Microservices provide a new perspective on reuse from SOA days?
  – SOA architectures were based on relatively “dumb” services and lots of orchestration
  – Think Choreography vs. orchestration
Case Study: **Principle 4 - Leverage “The Power Of Small” To Architect For Change**

The team’s objective is to design their architecture based on *smaller, loosely coupled components*. How would you help them do this?
Case Study: **Principle 4 - Leverage “The Power Of Small” To Architect For Change**

- Not using any of the *vendor specific extensions* to SQL would allow them to *swap one DBMS for another*, if they discover at some point in the life cycle of the “MobileShopping” system that they need scalability at a reasonable cost beyond what their initial choice of DBMS provides.

- Using small, loosely coupled components (and even micro-services if possible) allows them to *replace a component when necessary*, instead of attempting to enhance it and perhaps introducing some new defects.
CA Principle 5: **Architect for Build, Test and Deploy To Deliver Capabilities Continuously**

- The first 4 CA principles are not specific to Continuous Delivery
  - This changes with Principle 5!

- Optimize the architecture for *the whole SDLC* not just for the “design/build” phase of the process

- The Architect needs to take into account the following requirements:
  - *Integration*
  - *Testing*
  - *Deployment*
  - *Production support*

- Some of the following techniques can be leveraged:
  - Design small, API testable services
  - Service Virtualization
  - Any others?
Case Study: *Principle 5 - Architect for Build, Test and Deploy To Deliver Capabilities Continuously*

How could you help the team optimize their software delivery process?

From this:

To this:

Focus on the *software deployment* process and think *automation*
Case Study: Principle 5- Architect for Build, Test and Deploy To Deliver Capabilities Continuously

- The architect elects to use a container approach to facilitate deployment of the application.
- Using that approach, the application and its dependencies are packaged in a container.
- This allows applications and associated components to be rapidly deployed in multiple environments, including Cloud environments, both internal and external to the Enterprise.
CA Principle 6: *Model The Organization Of Your Teams After The Design Of The System To Promote Interoperability*

- So far we dealt with Process and Technology
  - What about *people*?

- Could the *organization* of your teams have an *impact* on your system *designs*?

- Organizing teams in *layers* does not promote collaboration. It creates *communication issues*!

- Conway’s Law (Melvin Conway – 1968): “Organizations which design systems... are *constrained* to produce *designs* which are copies of the *communication structures* of these organizations”

- Model the organization of your teams after your *desired architecture*!
Case Study: *Principle 6 - Model The Organization Of Your Teams After The Design Of The System To Promote Interoperability*

Architectures affect the organization, but *Conway’s Law* can also be used in reverse. It can either work in our favor when we model our teams after the design we would like to create and implement, or against us if a legacy team organization negatively impacts our design because of interoperability issues between the teams which eventually will result in multiple defects and associated rework.

How would you reorganize these teams to promote *interoperability*?
Case Study: **Principle 6 - Model The Organization Of Your Teams After The Design Of The System To Promote Interoperability**

- The team decides to organize themselves around the following capabilities that will be delivered by the new “MobileShopping” application:
  - **Capability 1**: Obtain quotes for a product – including competitors’ quotes
  - **Capability 2**: Place an order
  - **Capability 3**: Order Fulfillment

- Combined with *pairing*, this approach results in better knowledge of the system being delivered, better ownership and better *productivity*!
Some Thoughts on *Modernizing Monolithic Systems*

- So far we dealt with developing a brand new system of engagement
  - But what about enhancing an *existing monolith*?
  - Could we still use *Continuous Architecture*?

- Start on a small scale
  - Focus on areas that are likely to *change often*
  - Use *Principles 4, 5 & 6*

- Let’s return to the Case Study
  - We need to change the Quoting System!

© Pierre Pureur & Murat Erder
Modernizing Monolithic Systems – Brief Case Study

- Older, mainframe based, COBOL application developed a few decades ago
  - **Input/Output subsystem**: Reads/updates data from a DB2 database
  - **Data transformation subsystem**: formats/transforms data for use in the Quoting Engine. Heavily modified over time, very large (over a million lines!). Very unstable
  - **Quoting Engine subsystem**: Interfaces with a 3rd Party Quoting Engine

How would you add a new transformation for a new input variable?

Use the “Strangler” Pattern”. Assume that Java is available on the mainframe
What Exactly is *The Strangler Pattern*?

- The goal of this pattern is to *progressively replace* a legacy component with a new component.
- The process is as follows:
  1. Create a new component that *replaces a small percentage* of an existing system.
  2. Create an “*Abstraction Layer*” that invokes the new component instead of the legacy one.
  3. Repeat until the legacy component has been *eliminated*.

See [http://paulhammant.com/2013/07/14/legacy-application-strangulation-case-studies/](http://paulhammant.com/2013/07/14/legacy-application-strangulation-case-studies/)
The team *creates a new micro-service* written in Java since they have the ability to run Java on the mainframe.

They modify the existing application code *to invoke the new transformation micro-service* instead of the old Data Transformation sub-system when the new transformation is required.

They also obtain the agreement from the Quoting System application owner to “*freeze* the old Data Transformation sub-system”

This approach requires creating a *set of comprehensive regression tests* which must be automated.
Conclusion – Why Does Continuous Architecture Work?

- Leveraging the contents of our “Continuous Architecture” toolbox helps architects address and eliminate the bottlenecks that may be created by traditional architecture when attempting to support Agile projects.

- In addition, the Continuous Architecture approach speeds up the software development and delivery processes by systematically applying an architecture perspective and discipline.

- It supports our goal of delivering software at an ever increasing speed to create competitive differentiators.

- The Continuous Architecture approach works because we do not think of it as a formal methodology.
Questions?
For More Information

- Please check our blog at https://pgppgp.wordpress.com/

- Please also check our "Continuous Architecture" book (www.store.elsevier.com/9780128032848)
THANK YOU!