Emergent Architecture

Just Enough Just In Time
Emergent Architecture

With Scrum and other forms of agile software development we focus on incrementally evolving architecture one sprint or iteration at a time and avoid the potential waste of big design up front.

- What's this really mean?
- We need to do just enough just in time.
- We need to deliver a potentially shippable increment of working software every sprint.
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.
Agile Manifesto Principles

Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.

**Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.

**Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must **work together** daily throughout the project.

Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is **face to-face conversation**.

**Working software** is the primary measure of progress.

Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to **technical excellence** and good design enhances agility.

**Simplicity** -- the art of maximizing the amount of work not done -- is essential.

The best architectures, requirements, and designs emerge from **self-organizing teams**.

At regular intervals, the team **reflects** on how to become more effective, then tunes and adjusts its behavior accordingly.
The Wisdom of Building Roads and Bridges ...

- Building a Bridge
  - Design risks are low
  - Design costs are low relative to construction costs
  - Schedule and resource the tasks
  - Cannot deliver incrementally

- Building Software
  - More is unknown than known
  - Design risks are greater
  - Value-up measures value delivered incrementally
BDUF = Big Design Up Front

- The approach in which the design is to be completed and perfected before implementation starts
Why Big Design Up Front Seems Good

• Requirements are better understood
• Thinking things out ahead of time saves headaches
• Changes to specs are cheaper than changes to code
• Bugs are cheaper to fix early
Why Big Design Up Front Isn’t Good

- Requirements aren’t well known in the first place
- Requirements change
- Developers are not able to foresee all problems
  - Some implementation is required to flesh out the details
- Customers are not able to tell you what they want
  - Only what they don’t want … after they see it

There is no business value in architecture or design!
## Lean Architecture

<table>
<thead>
<tr>
<th>Lean Architecture</th>
<th>Classic Software Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defers engineering</td>
<td>Includes engineering</td>
</tr>
<tr>
<td>Gives craftsman “wiggle room” for change</td>
<td>Tries to limit large changes as “dangerous” (fear change?)</td>
</tr>
<tr>
<td>Defers implementation (delivers lightweight API’s and descriptions of relationships)</td>
<td>Includes much implementation (platforms, libraries) or none at all (documentation only)</td>
</tr>
<tr>
<td>Lightweight documentation</td>
<td>Documentation-focused, to describe the implementation or compensate for its absence</td>
</tr>
<tr>
<td>People</td>
<td>Tools and notations</td>
</tr>
<tr>
<td>Collective planning and cooperation</td>
<td>Specialized planning and control</td>
</tr>
<tr>
<td>End user mental model</td>
<td>Technical coupling and cohesion</td>
</tr>
</tbody>
</table>
What is *Emergent Architecture*?

- The opposite of Big Design Up Front
- It allows architecture to emerge as you develop
- Some architectural decisions will need to be made before you write your first line of code
  - Technologies, frameworks, languages, patterns, practices
- Most of these decisions will be hard to change later
Emergent Architecture

So where and when do we plan, and how much?
What about the big picture?
How does our architecture fit within the enterprise?
How does it facilitate our business objectives?
How do we manage risk?
And, what about all the details?
  • What tools are we using, what standards are we adhering to, how are we managing maintainability and all the other NFR's?
Is everything integrating together nicely?
And, what is role of the solution-enterprise architect in an agile world?
• It’s not impossible, but it isn’t easy either
• It takes smart experienced developers
  (and architects are developers, it’s all about the team)
Where are you today?

- Waterfall
- BDUF
- Some DUF
- Cowboy
- Emergent
So, who designed this thing anyway?
How much do you need ... and when?

You need to strike a balance, find the right time and effort for your project

- Some upfront planning is critical for large, complex projects
- Ongoing thinking, prototyping, and architecture experiments are important too

Choose the last responsible moment

- Just enough just in time
This has worked for me

• One day architecture session
  • Everyone with something to contribute in the room
  • Whiteboards that will persist for life of development
    • Update as needed
• What we need
  • Infrastructure
  • What we have to integrate with
  • How big is it?
• Works for greenfield & brownfield projects/products
Some Qualities of Agile Architecture

- Pragmatic, fit for purpose
- Modular
- Strive for simplicity
- No unintended redundancy
- No overlapping functionality
- Supports the important “ilities”
- Sustainable
- Provides business value
Fit for Purpose

• The solution should be suitable for the intended purpose
• Defined by title, user story, and acceptance criteria
• **Example**
  • Title: All cars must stop at One Microsoft Way
  • User Story: As the intersection safety officer I want all cars approaching One Microsoft Way to stop so no geek pedestrians get injured
  • Acceptance Criteria: solution should be universally understood, should be available 24x7, should have a low TCO
It’s all About Business Value

- The Scrum Guide says that every sprint must generate a “Done”, useable, and potentially releasable product increment.
- Therefore:
  - It is not ok to have an “architecture sprint”
  - It is not ok to have an “infrastructure sprint”
  - There is no such thing as sprint 0
  - And while we’re on the topic: there’s no such thing as “done done”
Business Value and Priority

• Let the customer value and prioritize the *what*, not the *how*

• Architecture and infrastructure decisions are the *how*
  • Architecture exists to serve the team, not the other way around

• Quality attributes (the “ilities”) become acceptance criteria or entries on the Definition of Done

• Let the team (not the customer) decide what is fit for purpose
Specifying Acceptance Criteria

This can be done incrementally

1. Initially, a story can just have a title
2. Later, the story can be updated to include a more detailed description
3. Later, the story can be updated to include detailed acceptance criteria
4. Later, the story can be linked to a test case with more detailed criteria
5. Later, the test case can have detailed, manual steps specified
6. Manual tests can be recorded and played-back as automated tests
Architecture is part of the backlog

- A good Scrum/Agile product backlog should include:
  - Features
  - Requirements
  - Enhancements
  - Bugs
  - Architecture
  - Knowledge acquisition
Agile Models

• Intended to communicate
• Minimal time invested
• Often informal
• Often discarded
• Little “m”
Risk

Types of Project Risks
• Schedule & budget
• Operational
  • execution
  • resources
  • communications
• Technical
  • too complex
  • poorly defined
  • misunderstood

Reduce, Mitigate Risk
• Develop incrementally
• Integrate often
• Inspect and adapt
• Design innovate if needed
Spikes

- A spike solution is a simple program to figure out answers to tough technical or design problems
- It only addresses the problem under examination and ignores all other concerns
- Most are not good enough to keep, expect it to be a throw away
- Typically within the time-box of a sprint
An Architect is part of the team

- No ivory towers
- Involved as a member of the development team
- Hands-on, do what’s needed
- Architectural leadership
- Mentors others
- Engaged with the business
Communities of Practice

• A group of people who share a craft and/or a profession
• Meet regularly, collaborate for integration and improvement
Developing architecture

• The team performs design as part of the sprint to provide a solid basis to support their sprint goal
• Some sprints (i.e. sprint 1) will be more design-heavy than others
  • The team should keep this in mind as they make their forecasts
  • Every sprint delivers potentially shippable increment(s) of functionality
Think in Slices Not Layers

- Layers = delayed value
- Slices = value every Sprint
Make Smart Decisions

• Leverage popular, maintainable, and testable architectures, frameworks, and patterns
• Build in the flexibility to adapt to reasonable changes without building a meta-product
• Don’t gold plate
Don’t Over-Architect the Product

- Build at least one increment of business functionality every sprint
- Build enough of the architecture and design to support that business functionality
- Build adequate architecture and design to meet any non-functional requirements

Solve today’s problem today, tomorrow’s problem tomorrow
Nature is not Linear

• There are boundaries where conditions change
• Boundaries likely have chaotic behavior
• Nature is not linear, nor is software
Your Architecture Will Change

• You’ll encounter boundaries
• Handle today’s problems today
  and tomorrow’s problems tomorrow
Minimize Documentation

• Create just enough documentation as a basis of discussion
  • Choose whiteboards over electronic media
• Models can help you
  • Understand and visualize a system, application, or component
• Models can hurt you
  • Don’t model what’s not in scope for that sprint
  • Don’t become a slave to the model
• Let the Team decide if and when to use models
  • Burn them at the earliest responsible time
When Should you Maintain Documentation?

• When there is business value in that documentation
  • i.e. an ISV providing API docs
• When the documentation can be auto-generated or serves additional technical purposes
  • i.e. generating Sequence diagrams
  • i.e. using Layer diagrams for validation
Layer Diagrams

- Layer diagrams allow you to visualize the logical architecture of your system.
- A layer diagram organizes the physical artifacts in your system into logical, abstract groups called layers.
- These layers help you identify, describe, and differentiate the kinds of tasks that those artifacts perform.
- More than just pretty pictures, these diagrams can be used for ongoing validation.
Effective Emergent Architecture

- Use Principles, Patterns, and Practices
- Refactor
- Test early, often, and automatically
- Application Lifecycle Management
- Deploy frequently
Principles, Patterns, and Practices

• Use popular Principles effectively
  • Coupling, cohesion, composition, encapsulation, ...

• Use popular Patterns effectively
  • GoF, .NET, Microsoft P&P, MVC, MVP, MVVM, ...

• Use popular Practices effectively

<table>
<thead>
<tr>
<th>Behavior-Driven Development (BDD)</th>
<th>Domain-Driven Design (DDD)</th>
<th>Liskov Substitution Principle (LSP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defensive Programming</td>
<td>Don’t Repeat Yourself (DRY)</td>
<td>Open-Closed Principle (OCP)</td>
</tr>
<tr>
<td>Dependency Injection (DI)</td>
<td>Interface Segregation Principle (ISP)</td>
<td>Separation of Concerns (SoC)</td>
</tr>
<tr>
<td>Dependency Inversion</td>
<td>Inversion of Control (IoC)</td>
<td>Single Responsibility</td>
</tr>
<tr>
<td>Design by Contract (DbC)</td>
<td>Principle of Least Privilege (PLP)</td>
<td>Test-Driven Development (TDD)</td>
</tr>
</tbody>
</table>

And my favorite: YAGNI (You Ain’t Gonna Need It)
Refactoring

• A disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior

• You should refactor ...
  • Little and often
  • At the unit as well as architecture level
  • Using automated unit tests, CI, and code coverage
  • Using automated regression tests
  • Using modern tools
Know when you are “Done”

- Done defines when an increment of product functionality is potentially shippable
- Definition of “Done”
  - A simple, auditable checklist owned by the team
  - It can be influenced by organizational standards and specific requirements of the product or release
- Can be maintained ...
  - In a document or wiki
  - In your ALM tooling
Example: A Simple Definition of “Done”

- Designed
- Refactored
- Coded
- No clever techniques
- Code review
- Design review
- Unit tested
- Functional tested
- Unit test harness

- User Acceptance tested
- Integration tested
- Regression tested
- Performance tested
- Security tested
Be Transparent

• Everyone should be able to inspect ...
  • The team’s definition of “Done”
  • The product backlog (user stories)
  • The sprint backlog (tasks)
  • The impediments
  • Deployments
  • Test results
  • Bugs

• Everyone should adapt to these inspections
Emergent Architecture is Important

The best architectures, requirements, and designs emerge from self-organizing teams.

Make it a part of your whole product and application lifecycle.
Burndown

- Deliver business value every Sprint
- Architects are active members of the development teams
- Leverage smart people and experience
- Big design up front is waste
- Delay requirements until last responsible moment
- Only keep documentation if it delivers business value
- Choose popular principles, patterns, practices
- Inspect and adapt every sprint

Emergent Architecture...
Just Enough, Just in Time
Learning More

1. *Just Enough Software Architecture: A Risk-Driven Approach* by George Fairbanks
3. *Software in 30 Days* by Ken Schwaber and Jeff Sutherland

These books provide valuable insights into software architecture and agile methodologies.
For More Info

- https://www.scrum.org/Resources/Nexus
- http://www.scaledagileframework.com/agile-architecture/
- http://www.mountaingoatsoftware.com/blog/agile-design-intentional-yet-emergent
- http://www.slideshare.net/rwirfs-brock/why-we-need-architects-and-architecture-on-agile-projects
Thank You!

Mike Vincent
MVA Software
mikey@mvasoftware.com
www.mvasoftware.net