Introducing Joseph

Founder and Architect, The Refactory, Inc.
Pattern enthusiast, author and Hillside Board President
Author of the Big Ball of Mud Pattern
Adaptive Systems expert (programs adaptive software, consults on adaptive architectures, author of adaptive architecture patterns, metadata maven, website: adaptiveobjectmodel.com)
Agile enthusiast and practitioner
Business owner (leads a world class development company)
Consults and trains top companies on design, refactoring, pragmatic testing
Amateur photographer, motorcycle enthusiast, enjoys dancing samba!!!
Loves Sushi, Ramen, Taiko Drums 😊
What is Software Engineering?

Software engineering (SE) is the application of engineering to the development of software in a systematic method....Wikipedia

Software Engineering

Definition of SOFTWARE ENGINEERING: a branch of computer science that deals with the design, implementation, and maintenance of complex computer programs

software engineer

noun

Webster’s Definition
Software Engineering

Definition of SOFTWARE ENGINEERING
1. **Development** of **procedures** and systematic **applications** that are used on electronic machines. Software engineering **incorporates** various accepted **methodologies** to **design** software...takes into consideration what **type of machine** the software will be used on, **how the software will work** with the machine, and what **elements** need to be put in place to **ensure reliability**.

2. **Higher education degree program**, which usually requires a certain number of courses to be completed in order to receive certification or a degree.

*Business Dictionary’s Definition*

---

Software Engineering

Definition of SOFTWARE ENGINEERING
: detailed **study** of **engineering** to the **design**, **development** and **maintenance** of software. Software engineering was introduced to **address** the issues of **low-quality** software projects. Problems arise when a software generally exceeds timelines, budgets, and reduced levels of quality. It **ensures** that the application is **built consistently, correctly, on time** and **on budget** and **within requirements**.

*Economic Times Definition*
Software Engineering

Definition of SOFTWARE ENGINEERING:
the process of analyzing user needs and designing, constructing, and testing end user applications that will satisfy these needs through use of software programming languages. It is the application of engineering principles to software development. In contrast to simple programming, software engineering is used for larger and more complex software systems, which are used as critical systems for businesses and organizations.

Techopedia’s Definition

Software Engineering

Definition of SOFTWARE ENGINEERING:
the application of principles used in the field of engineering, which usually deals with physical systems, to the design, development, testing, deployment and management of software systems. Uses a disciplined, structured approach to programming ... with the goal of improving the quality, time and budget efficiency, along with the assurance of structured testing and engineering certification.

TechTarget’s Definition
So Really... What is Software Engineering???

What about Agile?
architecture quality can be invisible

...especially when the spotlight is on

FEATURES
The Solution
What’s below the waterline?

all those “ilities” we can’t ignore

...
Agile Myths

- Simple solutions are always best
- We can easily adapt to changing requirements (new requirements)
- Scrum/TDD will ensure good Design/Architecture
- Good architecture simply emerges from “good” development practice
- You always go fast when doing agile
- Make significant architecture changes at the last moment

“www.agilemyths.com”

Big Ball of Mud

Alias: Shantytown, Spaghetti Code

A BIG BALL OF MUD is haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle.

The de-facto standard software architecture. Why is the gap between what we **preach** and what we **practice** so large?

We preach we want to build high quality systems but why are BBoMs so prevalent?
A BIG BALL OF MUD is haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle. The de facto standard software architecture. Why is the gap between what we preach and what we practice so large? We preach we want to build high quality systems but why are BBoMs so prevalent?

Maintaining code in a tidy state should be part of the work. “But it requires constant attention to this sort of hygiene,” Robert Chatley tells me when we meet at Imperial College.

Worse is Better

Ideas resembles Gabriel’s 1991 “Worse is Better”

Worse is Better is an argument to release early and then have the market help you design the final product…It is taken as the first published argument for open source, among other things

Do BBoM systems have a Quality?
What exactly do we mean by "Big"?

Well, for teams I consider > $10^2$ big and for code I consider > $10^5$ big

What is Large???

- $1,000,000$ (loc)
- $10,000,000$ (loc)
- $100,000,000$ (loc)
- $1,000,000,000$ (loc)
- Many terabytes of data
- Many dependencies
- Lot’s of connected pieces
- Many intertwined systems
Where Mud Comes From?

People Write Code → People make Mud

Keep it Working, Piecemeal Growth, Throwaway Code
Ultra-Large-Scale Systems

Ultra-large-scale (ULS) systems will be interdependent webs of software-intensive systems, people, policies, cultures, and economics…Cloud, IoT, Big data, …

http://www.sei.cmu.edu/uls/

What are Large Scale Systems

Large-scale systems include:

- Many lines of code (loc)
- Many dependencies
- Lots of stored data
- Lot’s of connected pieces
- Many intertwined systems
- Many overlapping policies
- Various security issues
- Many people involved
What are Large Scale Systems

Large-scale systems include:

- Very clusters of hardware
- Many networks integrated
- Lot’s of possible failure points
- Distributed Systems with multiple data centers around the world
- Systems that were not originally designed to work together
- No single team or timeframe

Need to Balance many forces

Simplicity
Scalability
Adaptability
Flexibility
Performance
Reliability
Features

...
Complex vs Complicated Systems (Cynefin Framework)

Enabling constraints
Loosely coupled
probe-sense-respond
Emergent Practice

Governing constraints
Tightly coupled
sense-analyse-respond
Good Practice

*Cynefin as of 1st June 2014* by Snowded - Own work. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Cynefin_as_of_1st_June_2014.png#/media/File:Cynefin_as_of_1st_June_2014.png

What do you Value?

TODAY
ANGER NOT
WORRY NOT
GRATEFUL BE
Diligent, in your endeavours, be
TO OTHERS, kindness show

How can I be more confident
Confidence

Values Drive Practices
Agile/Lean Design Values

- Core values:
  - Design Simplicity
  - Quick Feedback
  - Frequent Releases
  - Continuous Improvement
  - Teamwork/Trust
  - Satisfying stakeholder needs
  - **Building Quality Software**

- Keep Learning
- Sustainable Development
**Delivery Size???

incrementally

all at once

---

**Delivery Size is Key**

Large Delivery Size can cause many issues

**Issues:**
- More potential defects
- Longer time to get feedback
- Slower adjust time
- Harder to experiment
- Problems take a long time to fix
**Small Deliveries**

**Quick Feedback**

Master story list
1 day  Add user
2 days  Print itinerary
5 days  Cancel trip
3 days  Book permit
1 day   Update permit
...    ...
5 days  Create device
3 days  Add swap trade
100 days

---

**INTERNATIONAL MICROSERVICES CONFERENCE (IMC)**

IMC 2017 - Sydney, Australia 5-8 Nov 2017

**Microservices**

- Agility
- Scalability
- Resilience
- Testability
- Deployability
- Customization

*— Chris Richardson*
What about Quality?

Bad Code Smells

Have you ever looked at a piece of software that doesn't smell very nice?

A code smell is any symptom in the source code that can indicate a problem!
Neglect Is Contagious

- Disorder increases and software rots over time
- Don’t tolerate a broken window

http://www.pragmaticprogrammer.com/sppbook/extracts/no_broken_windows.html

Is it better to clean little by little?

Or to let dirt and mess accumulate?
Some dirt becomes very hard to clean if you do not clean it right away!

Technical Debt?

**Clean Code Doesn't Just Happen**

- You have to craft it
- You have to maintain it
- You have to make a professional commitment

“All fools can write code that a computer can understand. Good programmers write code that humans can understand.”

– Martin Fowler
But We Don’t Have Time!

Professional Responsibility

There’s no time to wash hands, get to the next patient!

Professionalism

Make it your responsibility to create software:
- Delivers business value
- Is clean
- Is tested
- Is simple
- Good design principles

When working with existing code:
- If you break it, you fix it
- You never make it worse than it was
- You always make it better

Refactoring

“If you value clean code…”
Refactorings

Behavior Preserving Program Transformations

- Rename Instance Variable
- Promote Method to Superclass
- Move Method to Component

Always done for a reason!!!

Refactoring is key and integral to most Agile processes!!!

---

Two Refactoring Types*

Floss Refactorings—frequent, small changes, intermingled with other programming (daily health)

Root canal refactorings—ininfrequent, protracted refactoring, during which programmers do nothing else (major repair)

* Emerson Murphy-Hill and Andrew Black in “Refactoring Tools: Fitness for Purpose”
Safe Refactorings

- Rename is always safe!!!
- New Abstract Class moving common features up
- Extract Method (always safe)
- Extract Interface / Extract Constant
- Pull Up / Push Down
- Create common component for shared internal methods
  - Fairly safe but can be harder to share

You Must Test

When you find smelly code, you often apply refactorings to clean your code.

Testing is a key principle for safe refactoring!
Common Wisdom

Work refactoring into your daily routine…

“In almost all cases, I’m opposed to setting aside time for refactoring. In my view refactoring is not an activity you set aside time to do. Refactoring is something you do all the time in little bursts.” — Martin Fowler

Strangler Pattern

Gradually create a new system around the edges of the old, letting it grow slowly over several years until the old system is strangled…

A natural wonder of the rain forests in Australia are the huge strangler vines. They seed in the upper branches of a fig tree and gradually work their way down the tree until they root in the soil. Over many years they grow into fantastic and beautiful shapes, meanwhile strangling and killing the tree that was their host.
PAUSE POINTS HELP

Kaizen

The Sino-Japanese word "kaizen" simply means "change for better", with no inherent meaning of either "continuous" or "philosophy" in Japanese dictionaries or in everyday use. The word refers to any improvement, one-time or continuous, large or small, in the same sense as the English word "improvement". (Wikipedia)

Most view it as Continuous Improvement…
Slack Time

Need Slack time to improve

Ways to get slack time…
- Monitor and Make Visible
- Reduce Waste (Muda)
- Inject time into process (retros, daily cleanup, …)

Try little experiments…

Continuous Improvement

“Retrospectives are Key!!!”

Small Steps we can take - next sprint!!!
Spotify: **Innovation**

**Regular Practices**

We must make time:
- Allocate time for dealing with tech debt
- Team building and education sessions
- Evaluate and Reflect…small changes
As we become more connected...

Large Scale SE Principles

- **Building Infrastructure**
  - Identify common problems, build infrastructure to address them
    - Important to not try to satisfy everyone
    - Perfection is the enemy of “Good Enough”
  - Don't build infrastructure just for its own sake
    - Identify common needs and address them
    - Don't imagine unlikely potential needs that aren't really there

- **Design for Growth**
  - Try to anticipate how requirements will evolve
  - Keep likely features in mind as you design base system
  - Think how design will scale if growth changes by 10X or 100X
Large Scale SE Principles

- **Design for Low Latency**
  - Low avg. times (happy users 😊) – 90% average idle time is ok
  - Lot's of caching and parallelism can be helpful

- **Make Applications Robust**
  - Aggressive load balancing
  - Failover to other replicas/datacenters
  - Bad backend detection: disable live requests until gets better
  - Do something reasonable even if not all is right
    - Better to give users limited functionality than an error page

- **Keep Software Clean**
  - Code reviews
  - Design reviews
  - Lots of testing
    - unit tests for individual modules
    - larger tests for whole systems
    - continuous testing system

---

**HOW SYSTEM QUALITY WORK CAN FIT INTO YOUR RHYTHMS**
“QUALITY IS NOT AN ACT, IT IS A HABIT.”
—ARISTOTLE

Some decisions are too important to leave until The Last Responsible Moment

SO

CHOOSE THE MOST RESPONSIBLE MOMENT
Qualify the Roadmap

“All you need is the plan, the roadmap, and the courage to press on to your destination”
— Earl Nightingale

Qualify the Roadmap

2017
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec 2018

- MOBILE WEB v1
- MOBILE WEB v2
- RICH MOBILE WEB APPS
- PC PLATFORM v1
- PC PLATFORM v2
- ONGOING RELEASES
- MOBILE RESEARCH
- ANDROID v1
- IOS v1
- RESPONSIVE DESIGN
- PERSISTENCE FRAMEWORK
- MOBILE GENERIC SERVICES
- SYBASE TO ORACLE MIGRATION
- LOAD BALANCING
- MOBILE SECURITY
- PLATFORM STABILITY
- CLOUD RESEARCH
- MICROSERVICES
- NO SQL / BIG DATA v1
- NO SQL / BIG DATA v2

DELIVERY

- TBD
- LOW RISK
- HIGH RISK
- NORMAL

RISKS

- TBD
- LOW RISK
- HIGH RISK
- NORMAL

ISSUES

- TBD
- LOW RISK
- HIGH RISK
- NORMAL

ON RADAR

- TBD
- LOW RISK
- HIGH RISK
- NORMAL
Qualify the Backlog

You can add backlog items for technical debt and quality-related architecture work… yes, you can

Make Architecture Work
Visible and Explicit

Positive Value
- Visible Feature
- Invisible Architectural Feature

Negative Value
- Visible Defect
- Technical Debt

Color your backlog—Phillipe Kruchten

http://philippe.kruchten.com/2013/12/11/the-missing-value-of-software-architecture/
How Quality Fits Into An Agile Process

Identify:
- Architecture Risks
- Key Quality Scenarios
- Landing Zone Criteria

Can Include Quality Items

Develop and Manage the Backlog

Incorporate Feedback

Quality Testing

Functional Acceptance Testing

Deploy to Stakeholders

Daily Review

Include relevant quality tasks

Run a Sprint

Plan a Sprint

Daily Review

Test Driven Development

TEST-FIRST DEVELOPMENT

- Write only enough code
- The test fails
- The test passes
- The test passes
- The test passes
- The test passes

REFACTORING

- Check if all the tests pass
- Update the failing tests
- Refactor some code
- Some tests fail
- The code quality satisfies
- Iterate

focus... Completion of the contract as defined by the test

focus... Alignment of the design with known needs
Mob Programming
A Whole Team Approach

Illustration © 2012 - Andrea Zuill

mobprogramming.org Twitter: @WoodyZuill

Large Scale Thinking

- Get feedback and advice early and often
  it is is ok to brainstorm and think
- Talk with colleagues and chat at a whiteboard
  Discuss designs & evaluate (spike solutions)
- Constantly monitor what is going on…
  Build operational dashboards and more
- Think carefully about interfaces
  (how will others use the interface)
  - Get feedback on your interfaces, evolve as needed
  - Learn from proven well-designed interfaces
Large Scale Practices

- Good Modularity and Abstraction principles still work
  - no one group (and no single timeframe) has created all the software, so do only what you can

- Be expansive in exception handling
  - When one happens, log all the relevant details; write the exception handler to try to repair the problems or at least continue in some fashion

- Log stuff just in case

- Write code to check consistency and validity of data, and run that code periodically or continuously in the background

- Write code to repair inconsistent or invalid data, preferably by reconstructing it from sources other than the bad data itself

- Don’t assume synchronization is perfect; tolerate messed up data

ONGOING QUALITY ACTIVITIES
Visibility is Important

Monitor System Qualities—Build An Operational Dashboard
Quality Focused Checklists

• Release Checklists*
  – Agreed upon checklist for quality and major architecture concerns

• Use at pause points
  – sprint planning, release planning, ...

*Thanks, James Thorpe for sharing your company’s checklist

Two Kinds of Checklists

1. Read-review
2. Do-confirm

---

Development Release Checklist

The code and architecture should be examined prior to release into our test environment. If any checklist cannot be checked, exceptions should be noted and communicated to the Product Owner and QA lead.

Code quality
- All code complies with the relevant coding standard.
- All code complies without any errors or warnings (full clean and build)
- Appropriate logging has been implemented throughout the code.
- All possible exceptions have been handled appropriately.
- The code has been checked for memory leaks.
- All tests and debug code have been removed.
- Code is appropriately documented.
- All dead code has been removed.
- All unit tests have been run without error.
- Unit tests have been written for all new code or code changes.

Architecture
- No web-service APIs have been created or modified without full documentation and architectural sign-off.
- No web-service data structures have been created or modified without full documentation and architectural sign-off.
- No database structures have been created or modified without full documentation and architectural sign-off.

Performance
- All web pages render in under 500 ms with a production workload
- All reports are generated in under 500 ms with a production workload
- No query takes more than 500 ms to return data with production data volumes.

Notes or Exceptions to the above.
Define Architecture Triggers

• Conditions that cause architecture investigation/ tasks
  – Quality target no longer met
  – Code quality metrics violations
  – ...
• Have broad system impact
Continuous Inspection

CODING DETECTION
METRICS (TEST COVERAGE, CYCLOMATIC COMPLEXITY, TECHNICAL DEBT, SIZES, …)
APPLICATION SECURITY CHECKS
ARCHITECTURAL CONFORMANCE

AUTOMATE WHERE YOU CAN!!!
Periodically Re-Evaluate Architecture Risks

Delivery and Feedback

Continuous Improvement

Implementation

Iteration Planning

Agile Values Can Drive Architectural Practices

- Do something. Don’t debate or discuss architecture too long
- Do something that buys you information
- Prove your architecture ideas
- Reduce risks
- Make it testable
- Prototype realistic scenarios that answer specific questions
- Incrementally refine your architecture
- Defer architectural decisions that don’t need to be immediately made
Patterns for Evolving Agile Architecture

USA PLoP 2015

Patterns for Evolving Agile Architecture

Asian PLoP 2015
Patterns for Being Agile at Quality

Core Patterns
- Breaking Down Barriers
- Integrate Quality

Identifying Qualities
- Finding the Qualities
- Agile Quality Scenarios
- Quality Stories
- Measureable System Qualities
- Fold-out Qualities
- Agile Landing Zone
- Recalibrate the Landing Zone
- Agree on Quality Targets

Making Qualities Visible
- System Quality
- Dashboard
- System Quality Radiator
- Qualify the Roadmap
- Qualify the Backlog
- Automate First
- Quality Checklists

Becoming Agile at Quality
- Whole Team
- Quality Focused Sprints
- Product Quality Champion
- Agile Quality Specialist
- Spread the Quality Workload
- Shadow the Quality Expert
- Pair with a Quality Advocate

QA to AQ
Patterns about transitioning from Quality Assurance to Agile Quality
Joseph W. Yoder¹, Rebecca Welke-Brack³, Adomar Aguiar¹
¹ The Refactory, Inc.
² Welke-Brack Associates, Inc.
3 joe@refactory.com, rebecca@quality-brack.com, adomaqaguiar@fe.up.pt

Abstract. As organizations transition from waterfall to agile processes, Quality Assurance (QA) activities and rules need to evolve. Traditionally, QA activities have occurred late in the process, after the software is fully functioning. As a consequence, QA departments have been “quality gatekeepers,” rather than actively engaged in the ongoing development and delivery of quality software. Agile teams incrementally deliver working software. Incrementated delivery provides an opportunity to engage in QA activities much earlier, ensuring that both functionality and important system qualities are addressed just in time, rather than too late. Agile teams embrace a “whole team” approach. Even though special skills may be required to perform certain development and Quality Assurance tasks, everyone on the team is focused on the delivery of quality software. This paper outlines 21 patterns for transitioning from a traditional QA practice to a more agile process. So of the patterns are completely presented that focus on where quality is addressed earlier in the process and QA plays a more integral role.

Patents to Develop and Evolve Architecture in an Agile Project, PLoP 2016,
Continuous Inspection, AsianPLoP 2016

...PATTERNS FOR TRANSITIONING FROM TRADITIONAL TO AGILE QA AND AGILE ARCHITECTURE

QA to AQ: Patterns about transitioning from Quality Assurance to Agile Quality, AsianPLoP 2014
QA to AQ Part Two: Shifting from Quality Assurance to Agile Quality, PLoP 2014
QA to AQ Part Three: Shifting from Quality Assurance to Agile Quality “Tearing Down the Walls”, SugarloafPloP 2014
QA to AQ Part Four: Shifting from Quality Assurance to Agile Quality “Prioritizing Qualities and Making them Visible”, PLoP 2015
QA to AQ Part Five: Being Agile At Quality “Growing Quality Awareness and Expertise”, AsianPLoP 2016
QA to AQ Part Six: Being Agile At Quality “Enabling and Infusing Quality”, AsianPLoP 2016

Copyright 2014 is held by the author(s). ACM 978-1-XXXX-XXXX-X.
Indicators You’ve Paid Enough Attention to Architecture

- Defects are localized
- Stable interfaces
- Consistency
- Developers can easily add new functionality
- New functionality doesn’t “break” existing architecture
- Few areas that developers avoid because they are too difficult to work in
- Able to incrementally integrate new functionality

Other Techniques for Improving Quality

Steve McConnell

Average is 40% for any one technique!
Combining techniques gives you quality (> 90%)
VALUES DRIVE PRACTICE
CALL TO ACTION

Manifesto for Agile Software Development

“We are uncovering easier ways of developing valuable products by doing it and helping others to do it. Through this work we have come to value:”

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.
Lazy Manifesto

“We are uncovering easier ways of developing valuable products by doing it and helping others to do it. Through this work we have come to value:”

- Keeping slack over being busy all the time
- Small high quality software over large complex software
- Doing only what is necessary over exhaustively discovering all tasks
- Doing less to deliver the same over doing more to deliver less

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

Relaxed Principles of Lazy Manifesto

“We follow these principles when they don’t add work:”

- Doing nothing is always an option.
- We seek to minimize the number of backlog items while keeping the value of the backlog.
- We believe to keep increasing velocity is not always good.
- We try to eliminate tasks that generate no value.
- We try to combine tasks to reduce latency and rework.
- We try to rearrange tasks to find problems early.
- We try to simplify all tasks as much as possible
- We are not afraid of eliminating our own tasks / processes by continuously acquiring new skills / capabilities.
- We expand capabilities over increasing capacities.
- We only work hard to make our work easier and safer.
- We always look to get help while we provide help to others with minimum effort.
- We never try to add an unnecessary principle simply to match with the other manifesto :)

[Image of Harada Kiro]
**Dogmatic**

**Synonyms:** bullheaded, dictative, doctrinaire, fanatical, intolerant

**Antonyms:** amenable, flexible, manageable

---

**Pragmatic**

**Synonyms:** common, commonsense, logical, practical, rational, realistic, sensible

**Antonyms:** idealistic, unrealistic

---

**Being Pragmatic**

- **Lot’s of Upfront Planning**
- **Lot of Design & Architecture**
- **Traditional or Waterfall**
  - **Rough Adaptive Plan (changing)**
  - **Right Balance of Design & Architecture**
  - **Being Agile**

- **No Planning**
- **No Design or Architecture**
- **Sometimes called Agile**

_Balance Between…_
It is a Journey

Commitment
Follow-through
Deliberate practices
Slack Time to Improve
Paying attention
Continuous Learning

Thanks!!!

joe@refactory.com
Twitter: @metayoda

www.joeyoder.com
www.refactory.com