Architecture in an Agile World

XP 2019 Workshop

(discussion of the balance between agility and architecture planning)

Dennis Mancl, dmancl -at- acm.org
Steven Fraser, sdfraser -at- acm.org
Werner Wild, piperlance0 -at- gmail.com

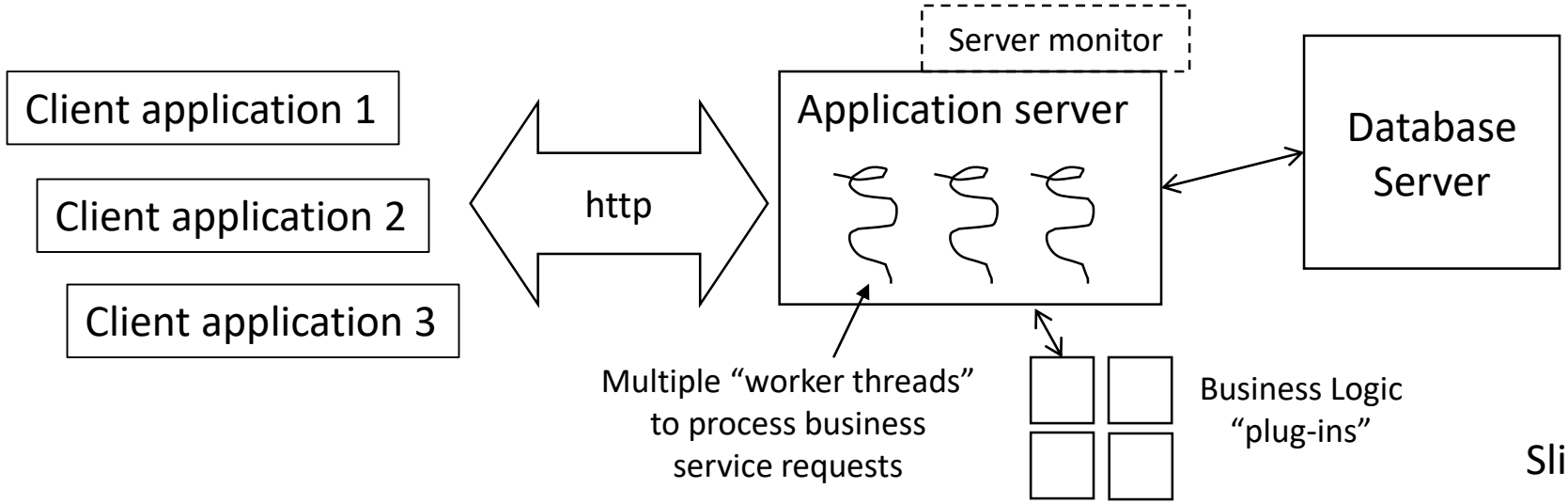
We live in an agile world

- Agile = requirements are always evolving
- How would we operate with fixed requirements?
  - Engineers might decide to follow a Roman model
  - “we want to build a road, bridge, aqueduct, building”
  - leaders decide on the route, capacity, size
  - we ask: “is there a standard plan that we can use?”
  - “don’t worry about future evolution”

- Consequences
  - Building cost might be high
  - Repair and maintenance costs are predictable
  - Problems when a city grows too big
  - Problems coping with barbarian invasion
- Problem: We need to think about “future requirements”

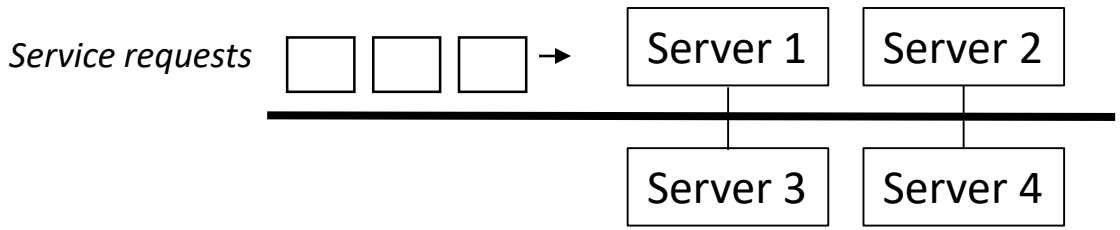
Architecture in an agile world

- What is architecture?
  - the overall structure that the design and code fits into
  - architecture defines the major components and interfaces
  - it is also the framework that ensures that the system meets certain critical properties (performance, security, reliability)
- The architecture may use existing components
  - Standard software libraries and frameworks
  - Build a flexible system – allow new modules to be added later



More about architecture

- Suppose we need high reliability
  - architecture might specify redundant data stores, multiple independent servers, and procedures for working around networking issues
  - the architect makes sure that the design will support a low failure rate, fast recovery after a failure, …
- Software experts talk about two kinds of architecture:
  - **System architecture** versus **software architecture**
  - it's mostly a distinction without a difference
  - some of the modules of a system may be hardware or human interactions
  - but we can model everything as if they were all software modules



More about architecture

- Architecture is part of the early design, but it isn't just one thing:
  - not just the high-level boxes and lines of the system
  - most good architectures have multiple levels... sometimes 5 or 6 levels are important
  - "a good architect is able to work in a complex environment... keeping at least 5 levels in his or her head"
- Always take economics into account
  - some architectures are based on "standards" (previous design experience, standard frameworks, programming language and messaging utilities)
- Why do we need architects? Experience counts
  - a good architect has seen weak and failing designs before

More about architecture

- Example: web apps…

Some "standard", some "customized" arch…

Network connections

Message formats

Transaction protocols

Business logic

Database access

Start by using standard tools and frameworks

- HTTP, SOAP, WSDL, transaction frameworks
- Java, PHP, Python, Ruby, JSON, Rails

Deployment environment

Software updates

Backups and recovery

Customization and localization

Use an environment like Tomcat to deploy all of the web services

- it is easy to add new services or update versions
- but it won't solve all of the problems...
  - what if a new component needs a few extra database tables?
  - or a version update adds some new database columns?
  - will you need a structured way to perform synchronized updates, schema migration, sanity checks, and security analysis??

Organizing development in an agile world

- What is agility?
  - software product development in small teams and short iterations
  - as much as possible, you have a partially working system early, and the system evolves to add more functionality
  - each cycle (iteration, Sprint) includes several activities

- **Requirements** – update the requirements based on customer input
- **Specifications** – for each work item
- **Software design** – for new packages or classes
- **Coding** – adding and modifying several code modules
- **Testing** – make sure the new code works
- **Testing** – make sure the old code hasn't been broken
- **Deploy** updates to the customer (or at least to a test lab)

How long is an iteration?

- Usually 2 to 4 weeks

Ugggh… How is my team going to do all of this work in a single iteration??

- Small steps – pick a small subset of the requirements, use simple design, write automated tests

More about agility

- The value of being agile:
  - it is easier to adapt to change (new customer requirements, new hardware, new interfaces)
  - better feedback from customers – really good for fixing minor problems that make our user interfaces annoying and inefficient
  - agility builds good teamwork
- Important: agile != hacking
  - not just a bunch of undisciplined developers doing what they want
  - most good agile methods have a lightweight planning cycle – using customer requirements and user stories to build an iteration plan
  - new code must be tested – and there is a big emphasis on building automated tests as part of the code base
  - some teams even use “test driven development” (writing tests before writing code)

More about agility

- Examples of popular agile methods
  - Extreme Programming (XP)
  - Scrum
  - Kanban (incorporates ideas from Lean software development)
  - each methodology has specifics on how to build teams, how to do iteration planning and execution, and what kinds of activities are essential
- An agile project is an “ecosystem that delivers software”
  - a single team or a group of teams
  - software products that are tested and good quality

More about agility

- Agile teams have a degree of "self-management"
  - instead of a manager handing out individual development tasks…
  - the team works together to break up the work -- and it is OK to work in pairs or small subgroups on complex tasks
  - not just "developers" – most team members wear multiple hats
  - one side benefit: you might not need as much written design documentation
  - in some agile methods (such as XP), big documents are seen as wasteful – you should be writing more code...

- Agile principles say:

  Avoid "Big Design Up Front"

- Architecture principles say:

  Establish a solid architecture early

- Is there a conflict??

What do you think??

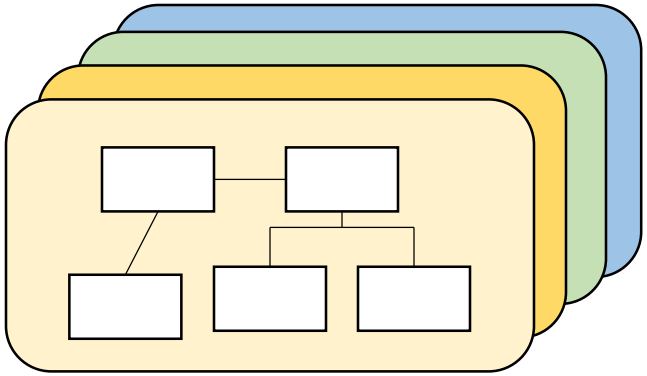
- What are the biggest architecture-related challenges faced by agile projects and teams?
  - Staff… not enough people with architecture knowledge and experience
  - Time… lots of pressure to deliver features, not enough time to do some architecture exploration
  - Legacy… our products might be built on unsteady foundations
  - Chaos… the requirements and the code base are evolving too quickly
  - [Add your favorite challenges here…]
  -
  -
  -
  -
- Do we have any good advice?? (a few ideas on pages 12-23)

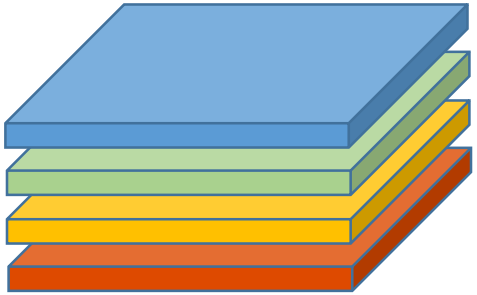
Some advice

- Several key messages about agile and architecture:
  - A **mixed model** of agile and architecture-driven development is a good way to make software product development effective.
  - A very important part of "being agile" is to **have respect for other people**, even if they aren't part of the inner team of agile developers.
    - *Agile is not just a hacker's club… you need to work with customers and stakeholders (including your own management)*
  - An very important part of "being a good architect" is to **advocate for the architecture** – working hand-in-hand with the developers to maintain a strong software structure.
    - *A big challenge… if the architects are outside of the agile team*
  - Use timeboxing and pageboxing to keep the architecture from becoming a "big design up front".
  - Everyone, not just architects, should be on the lookout for bad architecture smells – such as tangled code, missing -ility requirements, unbalanced test suites, and duplication.

Agilist myths about architects

- Agilists feel that architects are “always trying to push everyone around”
- In addition, agilists feel that architects often “overdesign”
  - These can be valid complaints. Architects really do try to control things too much – and an agile viewpoint could really help architects do their job better.





Too many diagrams?? Too many architectural layers??

Lightweight documentation: Show the “essential” parts of the design… a small group of scenarios, diagrams, interfaces, layers

Architects problems with agilists

- Architects often complain about agile teams – some issues related to lack of forward vision and feedback:
- Architecture is “paving some of the road ahead of you”
  - agilists sometimes forget to look to the future
  - if the design vision only extends to the end of the next iteration, it might be easy for the design to fall off a cliff
- Sometimes agile teams fail to get adequate feedback from stakeholders and customers
  - it isn’t always the agile team’s fault – but teams should always try to do better

We are halfway through the talk… we have heard all the problems, what can we do about them??

Architects

Even if you are an amateur architect, you can focus on the architecture skills you need to do better…

- What makes a good architect?
  - Navigator
  - Coach/teacher
  - Producer (like a "movie producer" – makes everything work for the production)
  - Seasoned veteran
  - Long-bearded wise man
  - Evangelist for the architecture
  - Student of the problem domain (always learning about the problem area)

Anti-patterns for architects:

- Diva (short-tempered and high-maintenance)
- Magician
- Over-accommodating ("sure, I can change that for you")
- Chess master (he/she is always moving the pieces around)

Agilists

Every agile team member has to be more than just a "programmer"… the team needs you.

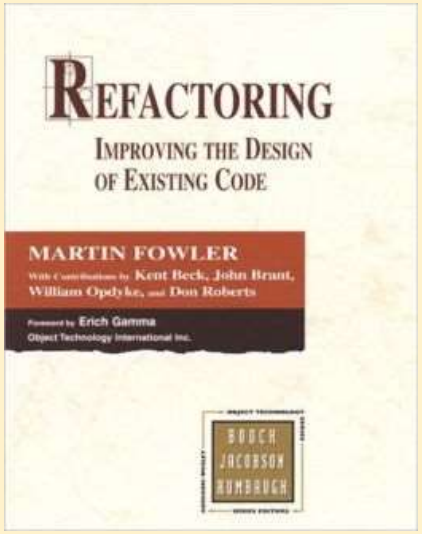
- What makes a good agilist?
  - Respects people (first and foremost)
  - Self-reflective (learns from experience)
  - Multi-lingual (in two senses: can work in multiple programming languages and converse with stakeholders and team members in their own terms)
    - good social skills (keeps on good relations with teammates)
  - Follows the values in the Agile Manifesto
  - Change advocate
  - Takes responsibility for his/her actions

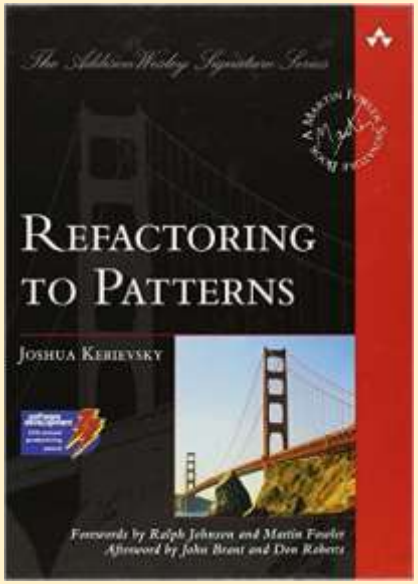
Anti-patterns for agilists:

- Oversimplifies previously learned wisdom
- Passive/aggressive agilist (skips many key agile practices)
  - only applies agile practices and values within his/her comfort zone
  - uses agile as an excuse for not being disciplined ("coding cowboy camouflage")

Refactoring

- A *refactoring* is a behavior-preserving program restructuring that can improve the design of software and support evolution and reuse.
- There are “small refactorings”
  - Clean up the code base – make sure that the code is readable; meaningful function names; simple data structures; easy to add new functionality within the existing code base
- Some refactoring work is “re-architecting”
  - Adjust the architecture for a reason:
    - Separate complex functionality into multiple modules
    - Add support for “reliable” behavior (multiple active objects, monitoring, transaction logs, recovery, restart)
  - “Wrap” some old code to make it reusable in a new system





Architect in an agile project – part 1

- Balance YAGNI and long-term / business value
  - YAGNI = “you aren’t going to need it”
  - In XP, the focus is to design and implement based only on the items in the current iteration, "do the simplest thing possible"
  - An architect can be the person who says “when you do not do the simplest thing”
- Work across multiple levels of abstraction
  - Provide “coaching” to team members who need to understand how low-level design is connected to the big picture
- Set up the architecture infrastructure
  - Architecture infrastructure might be the items the team identifies in “iteration 0”, or it might be a series of 3 or 4 exploratory early iterations to stabilize the initial architecture.
  - Philippe Kruchten – he advocates having some “architecture stories” in the backlog
  - “What Colour is Your Backlog”
  - https://www.youtube.com/watch?v=XWyBkmvNGrk

Architect in an agile project – part 2

- Architect communicates with customers and stakeholders (such as the project sponsor)
  - Direct communication between customers and all team members is good
  - Architects need to communicate with customers to understand requirements that go beyond the normal "functional requirements"
- Architect sets up guide rails – these may be some of the architecture decisions that guide future design-level decisions
- Architect advises people making priority decisions (so they are more aware of the future)
  - Product Owners must understand key architecture issues
  - Learn which architecture choices might block future features
  - The implementation of some stories might have positive or negative impact on performance, scalability, reliability,...
  - Refactor/reengineer: dedicate certain iterations to do refactoring work – architects might help recommend when a refactoring iteration is necessary.
- Architect "combats the forces of evil" – the outsiders who are trying to sabotage the agile process

Architecture smells – part 1

- Code that nobody wants to change
  - they are afraid to change it because it is complex or cryptic, and they don't want to break anything
- Missing metaphor (the team and customers need a common shared understanding and vocabulary about the problem)
- Overreliance on "refactoring iterations" to fix technical debt
- Abstractions that are hard to understand or communicate
- Missing -ility stories / tests / strategies
- Untestable code
- Unbalanced test suite (too many unit tests, not enough functional tests)

Why smells?? We do some refactoring when we run into "bad code smells" – we may need to do some architecture planning when we encounter bad architecture smells…

Architecture smells – part 2

- Test churn
- Problems in design evolution – the problem is sometimes called “you can’t get there from here... you have a design that is too rigid
- Inconsistent use of infrastructure
  - For example, there are some modules that use a standard messaging component while other components use some special local messaging code.
- Missing boundary stories
  - How does your system interact with other systems? What end-to-end stories do you need to write to understand the interface requirements between your system and external systems?
- Duplication without reason
  - Examples: using two different databases, multiple interface mechanisms

Some advice

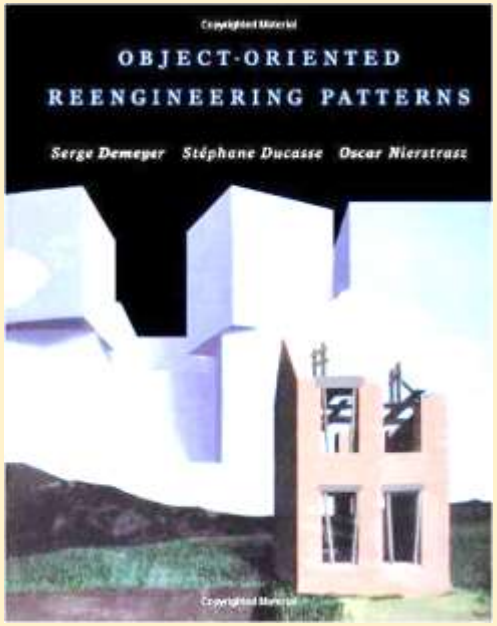
- Mixed model – don't be "pure agile"
  - Include architecture items in the agile planning
  - Make sure you have some architecture expertise on the team
- Respect for other people
  - Agile teams are more effective if there are external collaborations
  - Don't let the special "agile team" status become an obstacle
- Architect's role: advocate for the architecture
  - Don't let the code become too undisciplined
  - It is OK to take on some temporary technical debt
  - But in the long run, the code will be better if developers and architects work hand-in-hand

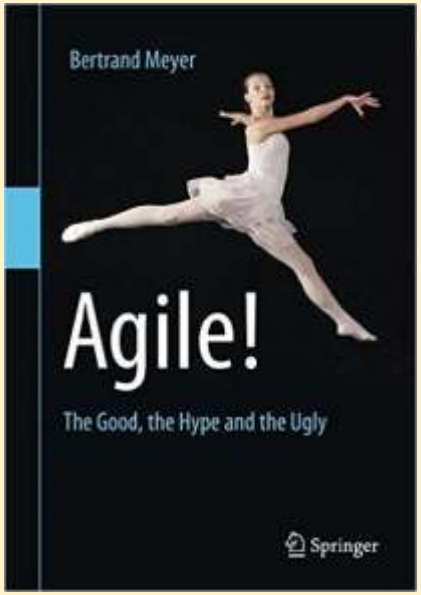
Some advice

- Big Design Up Front can be a problem
  - Use timeboxing and pageboxing to keep the architecture from becoming a “big design up front”
  - What does this mean? Set yourself a “budget” of time and effort on architecture planning
  - Divide the effort between functional and non-functional…
    - document some “key scenarios” (functional requirements)
    - show how the architecture addresses the “-ilities” (such as reliability, usability, portability, flexibility, performance, security)
- Look out for architecture smells
  - Tangled code, missing -ility requirements, unbalanced test suites, and duplication
  - It is an indication that you are “underinvested” in architecture
  - Everyone, not just architects, should be on the lookout for bad architecture smells

To learn more

- Object-oriented Reengineering Patterns
  - This book gives some good advice on reengineering
  - This book is “open source” – http://www.iam.unibe.ch/~scg/OORP
- Agile! The Good, the Hype, and the Ugly
  - Identifies the best and worst agile practices, how to do effective design in an agile context
  - Bertrand Meyer talk: https://www.youtube.com/watch?v=ffkIQrq-m34





To learn more

- Just Enough Software Architecture
  - Good tips – how to get a positive impact from a small amount of work on architecture planning
  - Focus architecture work to address “risks”
  - “Architecture Haiku”
  - http://georgefairbanks.com/software-architecture/architecture-haiku/
- Organizational Patterns of Agile Software Development
  - Good management book – also see web article
  - https://sites.google.com/a/gertrudandcope.com/info/Publications/Patterns/Process/OrganizationalPatterns.pdf