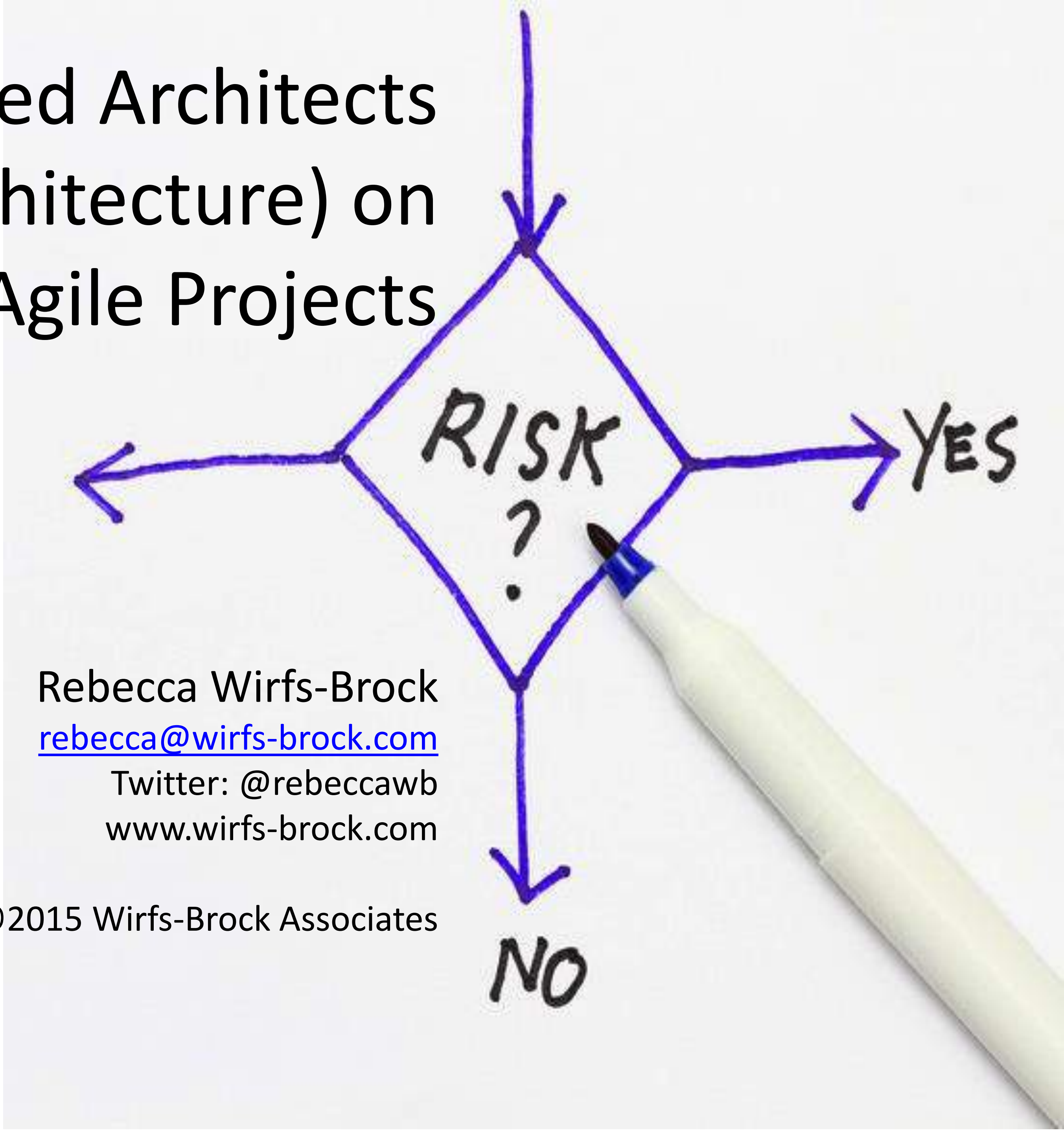
Why We Need Architects (and Architecture) on Agile Projects



Rebecca Wirfs-Brock
rebecca@wirfs-brock.com
Twitter: @rebeccawb
www.wirfs-brock.com

Three Questions…

? ? ?

- What is the role of an agile architect?
- How much architecting do you need and when?
- How can you manage architecture risk on small as well as large, complex projects?

Astronaut Architect?

Seagull Architect?

Infrastructure Freak?

Incompetent Fools?

Agile Design Values

Core values

- ✓ Design Simplicity
- ✓ Sustainable systems
- ✓ Continuous improvement
- ✓ Teamwork
- ✓ Communication
- ✓ Trust
- ✓ Satisfying stakeholder needs

Constant learning

Some Agile Myths

- Simple solutions are always best
- Building in flexibility is over-engineering
- We don’t need specialists (or architects)!
- We’re agile so we can adapt to any new requirement
- Don’t worry about *architecture*

Wayfinder Architect

Scouting—looking enough ahead

Active, integrative

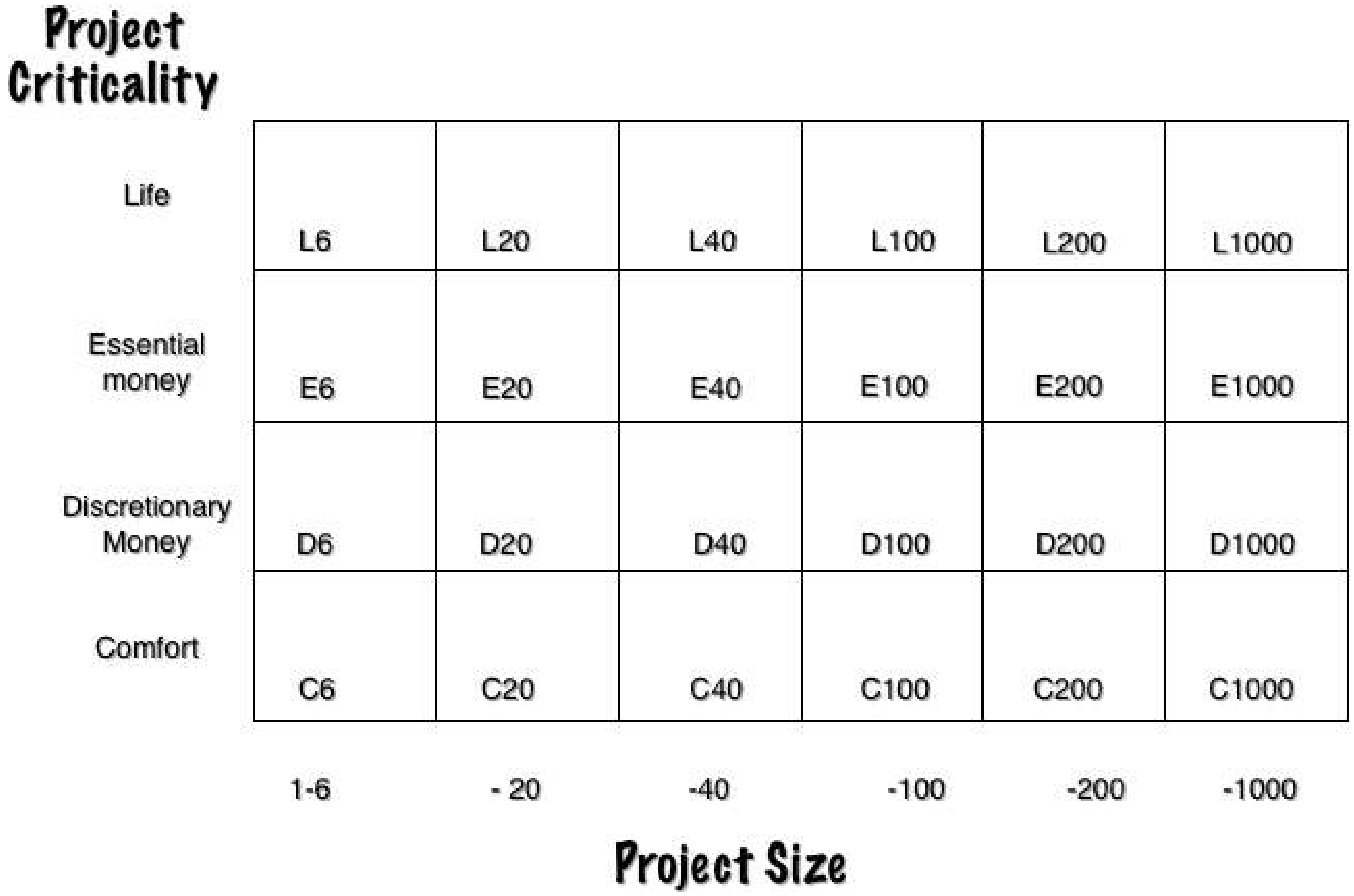
Exploring options

Steward Architect

Stewardship

- Sustainability
- Follow through
- Ongoing attention to little things that undermine the ability to grow, change and adapt
- Making difficult or tedious tasks easier

How Much Architecting Do You Need?



Project Criticality

Life	L6	L20	L40	L100	L200	L1000
Essential money	E6	E20	E40	E100	E200	E1000
Discretionary Money	D6	D20	D40	D100	D200	D1000
Comfort	C6	C20	C40	C100	C200	C1000
	1-6	- 20	-40	-100	-200	-1000

Project Size

Alistair Cockburn’s project characteristics grid

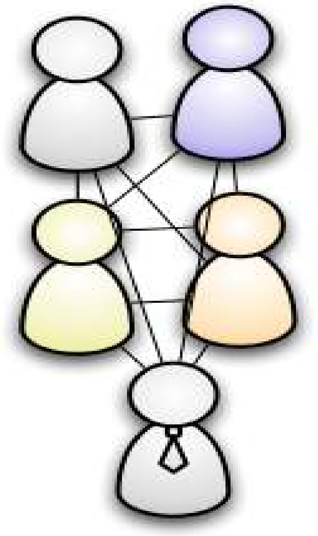
Qualities of Any Good Architecture

- Pragmatic. Does what it needs to without extras
- Designed for test
- Modular
- No unintentional data redundancy or overlapping functionality
- Supports performance, reliability, modifiability, usability,….

Small v. Large Projects

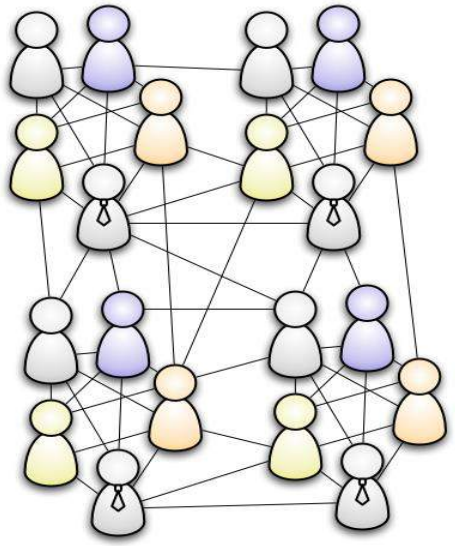
Small Projects

- 6-8
- non-life critical
- architecture often evolves OK without extra attention



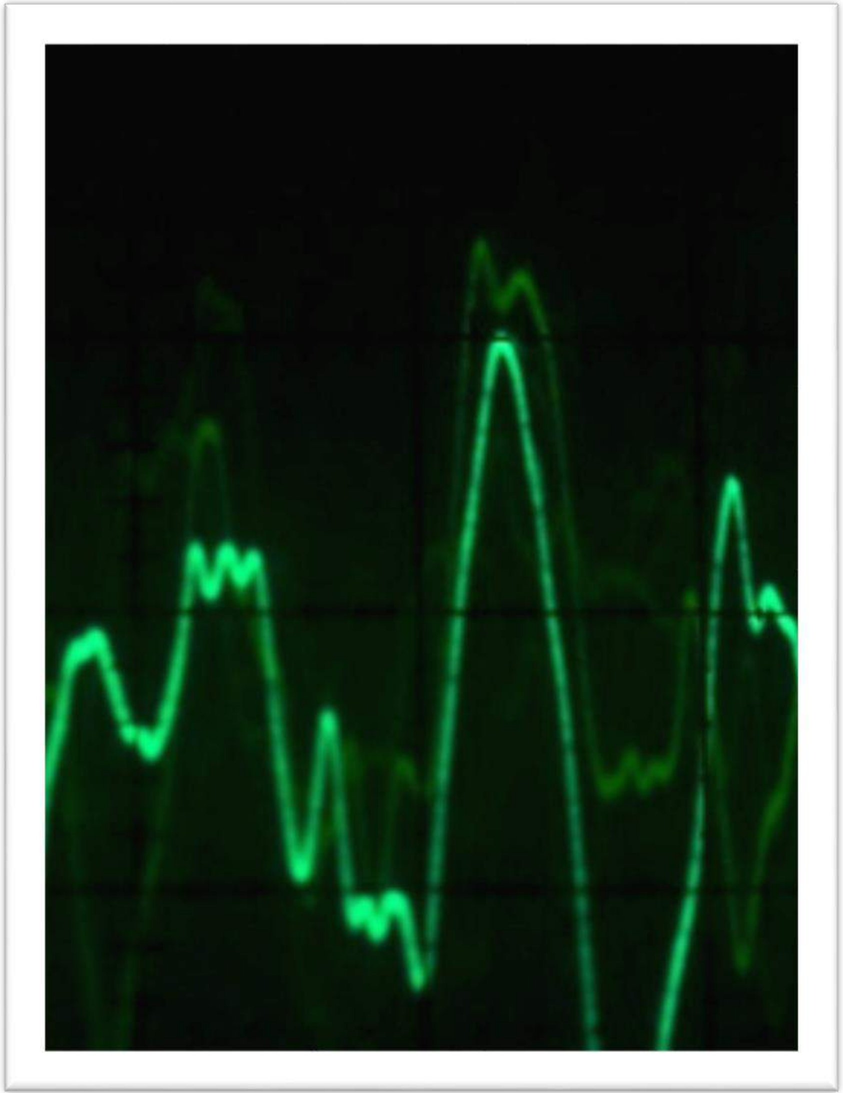
Large Projects

- Multiple teams
- "Naturally" emerging architecture can reflect organization structure
- Significant risks, challenges, unknowns, coordination



Small Project Architecture Practices: Design “Spikes”

- Goal: Figure out a design approach
- Time: Few hours to a few days
- Tools: CRC Cards, exploratory coding, whiteboard sketching

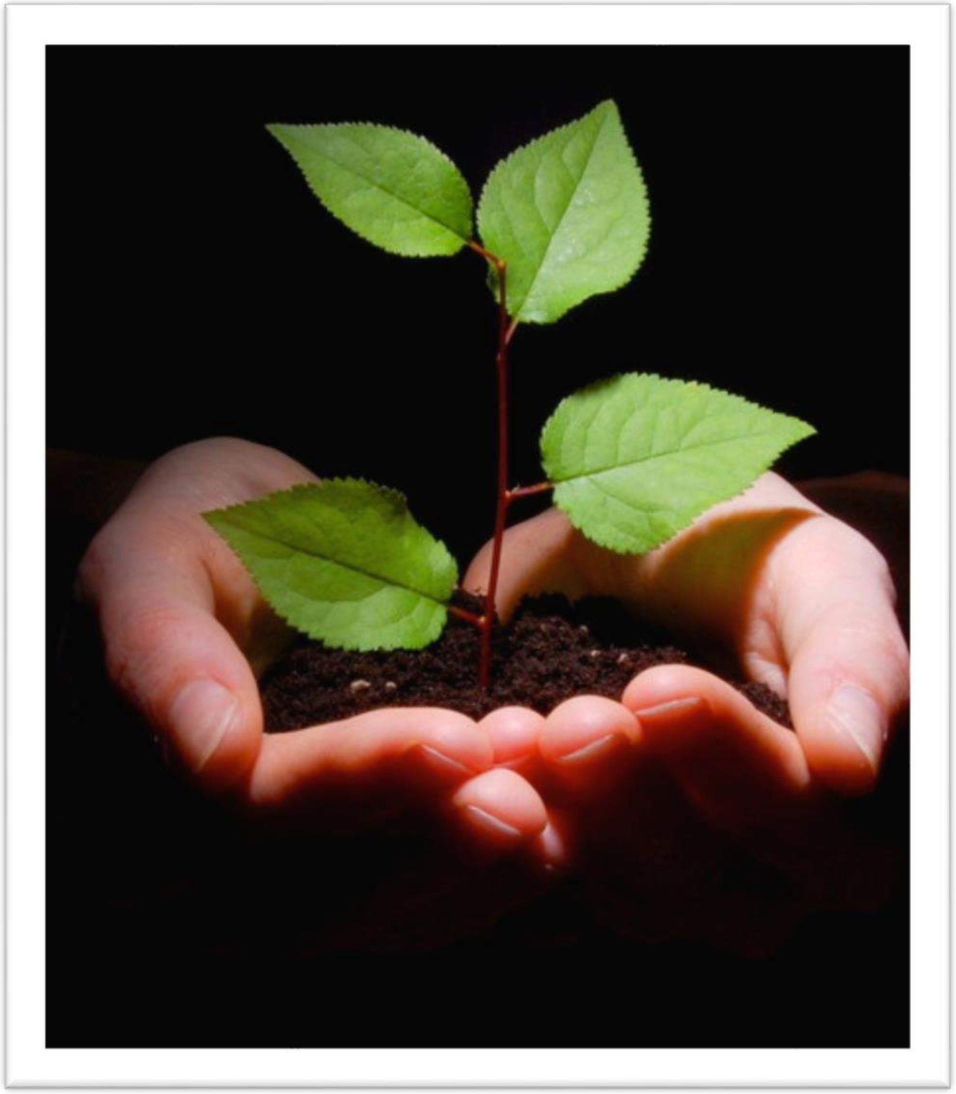


Small Project Architecture Practices: Experiment on Branches

- Goal: Experiment away from main code branch
- Time: Few hours to a few days
- When done: Merge or throwaway branch code

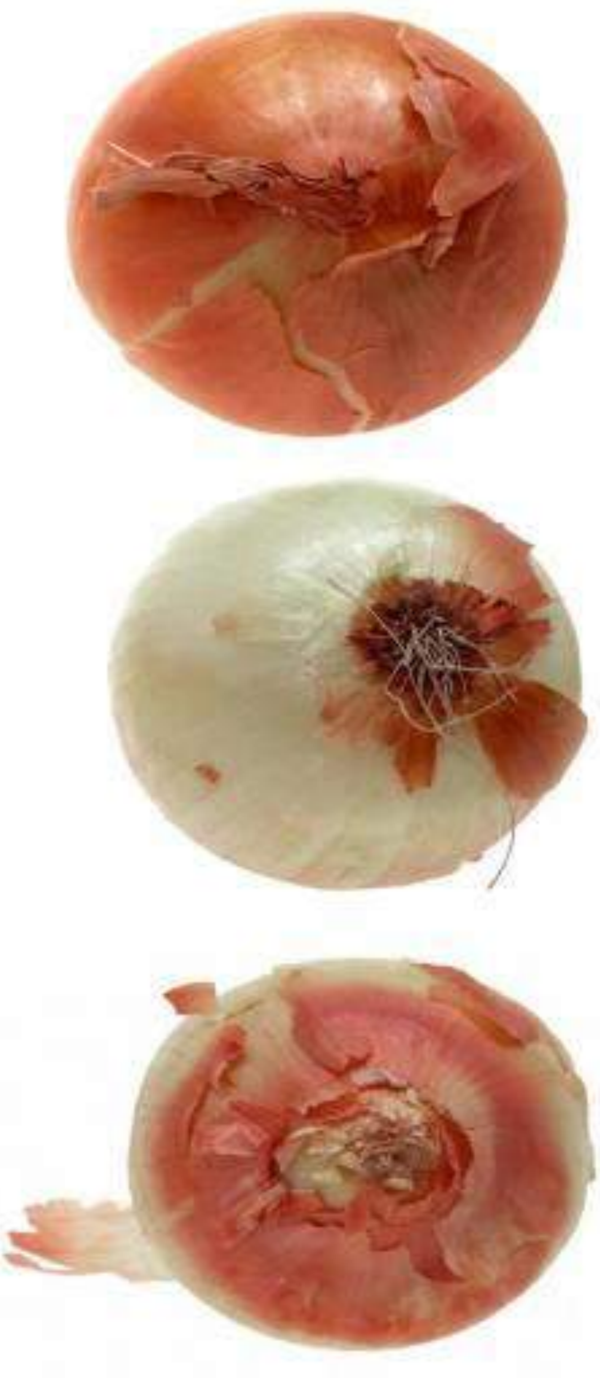
Small Project Architecture Practices: Incrementally Refine Abstractions

- Goal: Refactor to eliminate redundant code
- Time: Few minutes
- When done: Whenever you spot duplication



Small Project Architecture Practices: Manage Technical Debt

- Term invented by Ward Cunningham
- Piles up when you continually implement without going back to reflect new understanding
- Can have long term costs and consequences



All Tasks Aren’t Alike

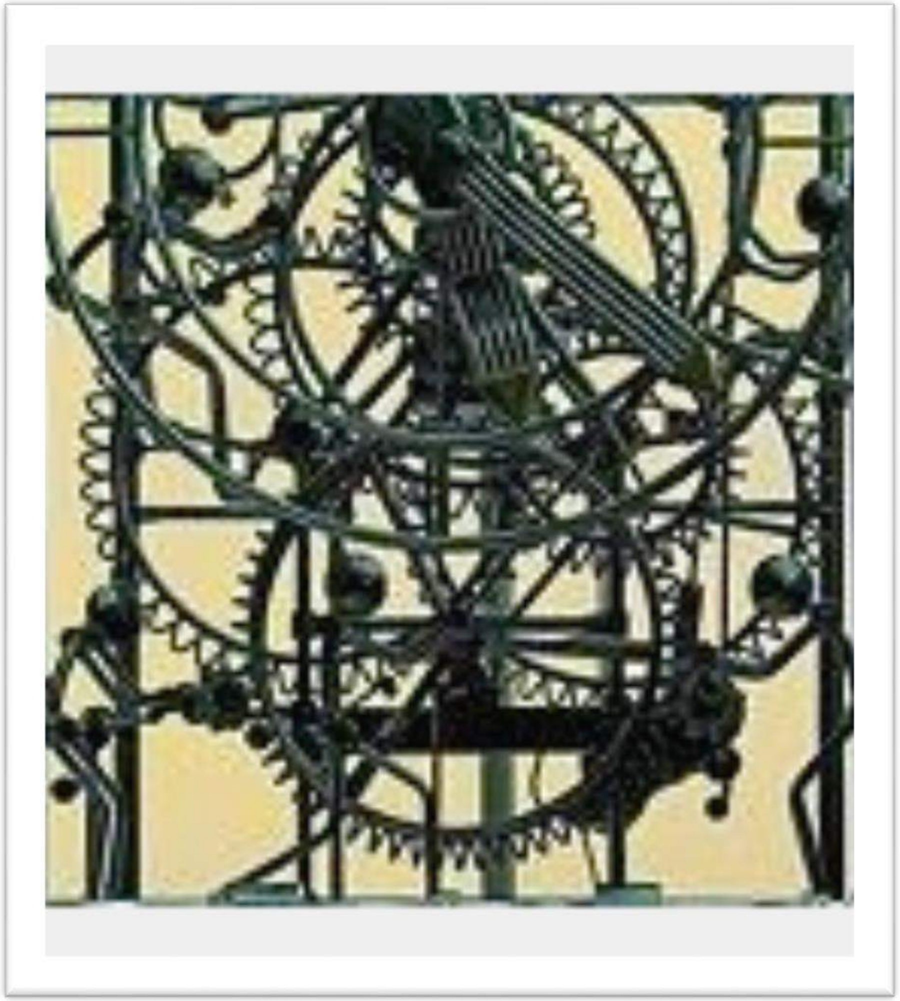
- **The Core—**fundamental to your software’s success
- **The rest—**requires far less creativity or inspiration
- **The Revealing—**lead to new, deeper understanding

Keep Architecture in Mind

- Sort tasks into “problem buckets”: core and the rest
- Make sure each iteration gets enough core work accomplished
- Get team involved on core issues
- Use post-iteration reflections to ask why things were harder

Architectural Practice: Reduce Technical Debt

- Integrate new learning into your code
  - Refactoring
  - Redesign
  - Rework
  - Code clean up
- Unit tests (functionality)
- Tests for architectural qualities (performance, reliability,…)



Architecture Practice: Sustainable Development

- Pay attention to architecture. Not only feature implementation
- Design consistency. “This is how we do x.”
  - Coding standards
  - Consistency (API use, errors, logging…)
- Stewards for architecturally critical code areas

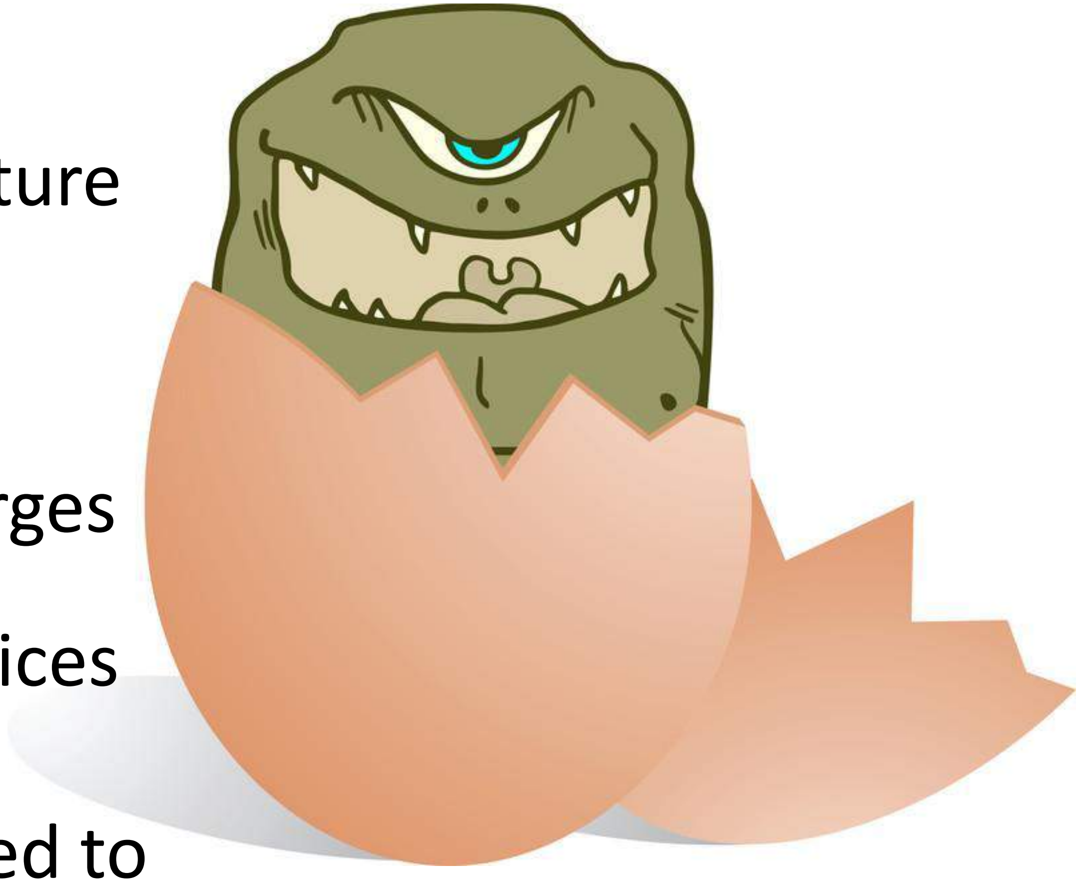
The Bigger the Project….

THE MORE THERE IS TO CONSIDER

Being Agile Does Not Guarantee

- You can make significant architecture changes at the last moment
- Good architecture automatically emerges from “good” development practices

- Sometimes you need to do more



Strike a Balance

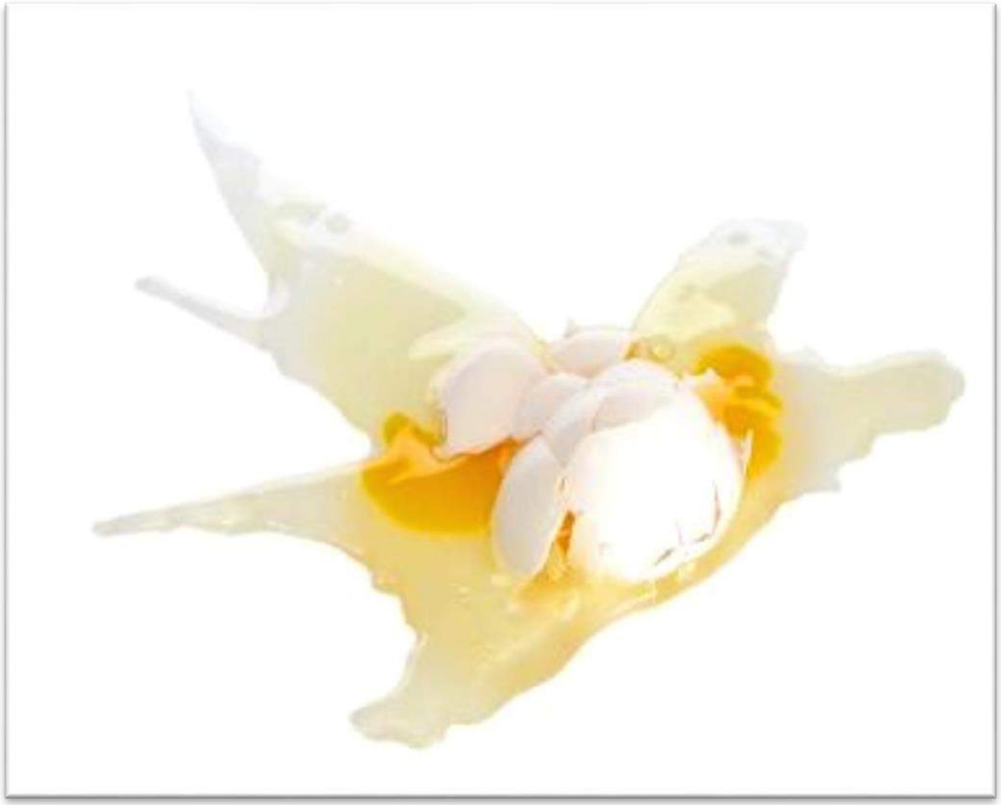
Some decisions are too important to leave until The Last Responsible Moment

so

CHOOSE THE MOST RESPONSIBLE MOMENT

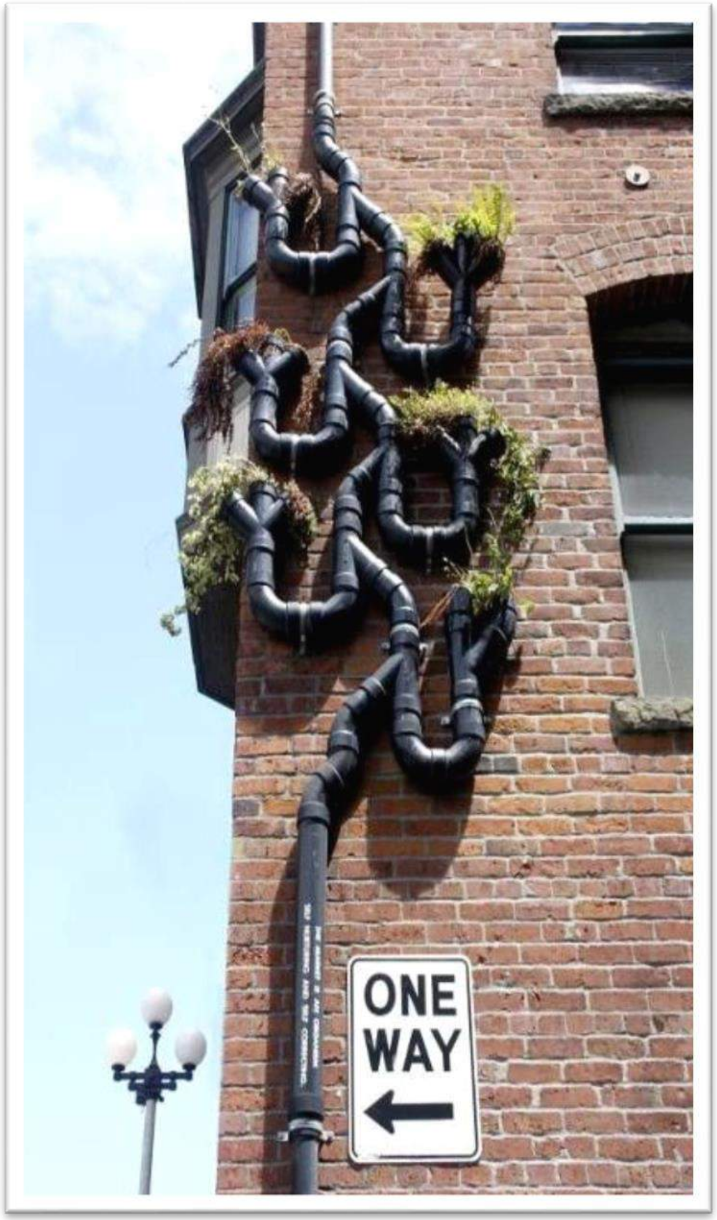
Types of Project Risks

- Schedule & budget
- Operational
  - execution
  - resources
  - communications
- **Technical**
  - **too complex**
  - **poorly defined**
  - **misunderstood**



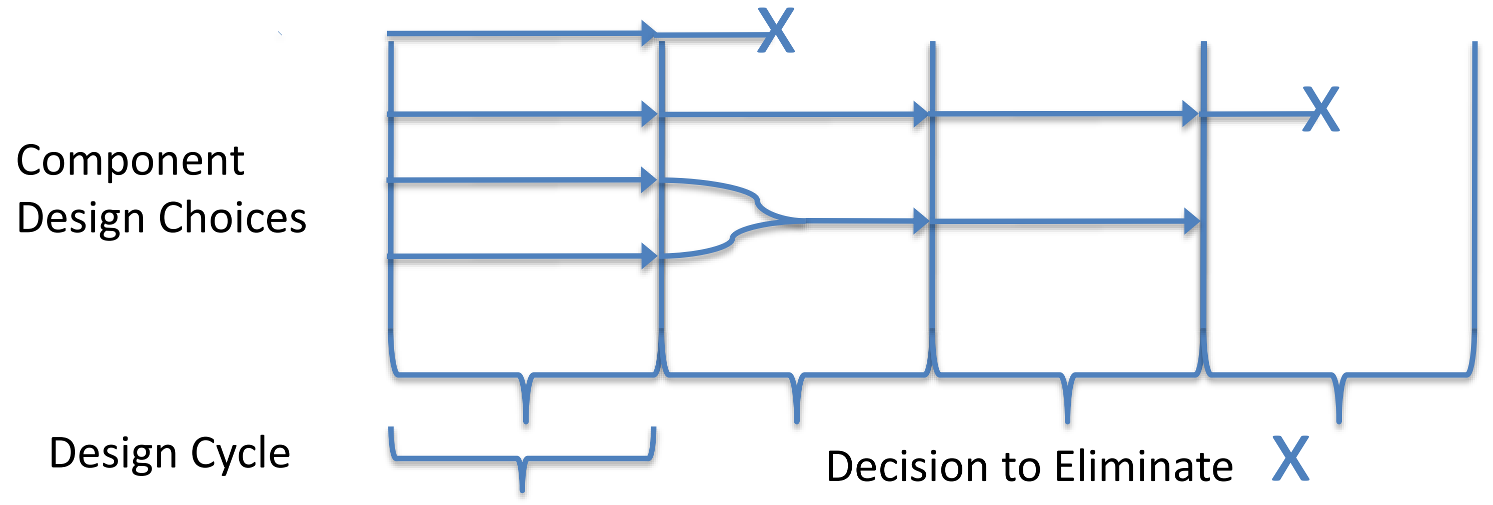
Architecture Debt

- Compromises in the system that have significant impacts
- Not isolated
- Costly to reverse
- Examples:
  - ignoring scalability
  - poor framework choices
  - inconsistent service interfaces



Additional Architecture Risk Reduction Tools for Larger Projects and Programs

- Grooming and vetting project/product road maps and timelines
- Landing zones
- Architecture spikes
- Risk reduction backlogs
- Set-based design

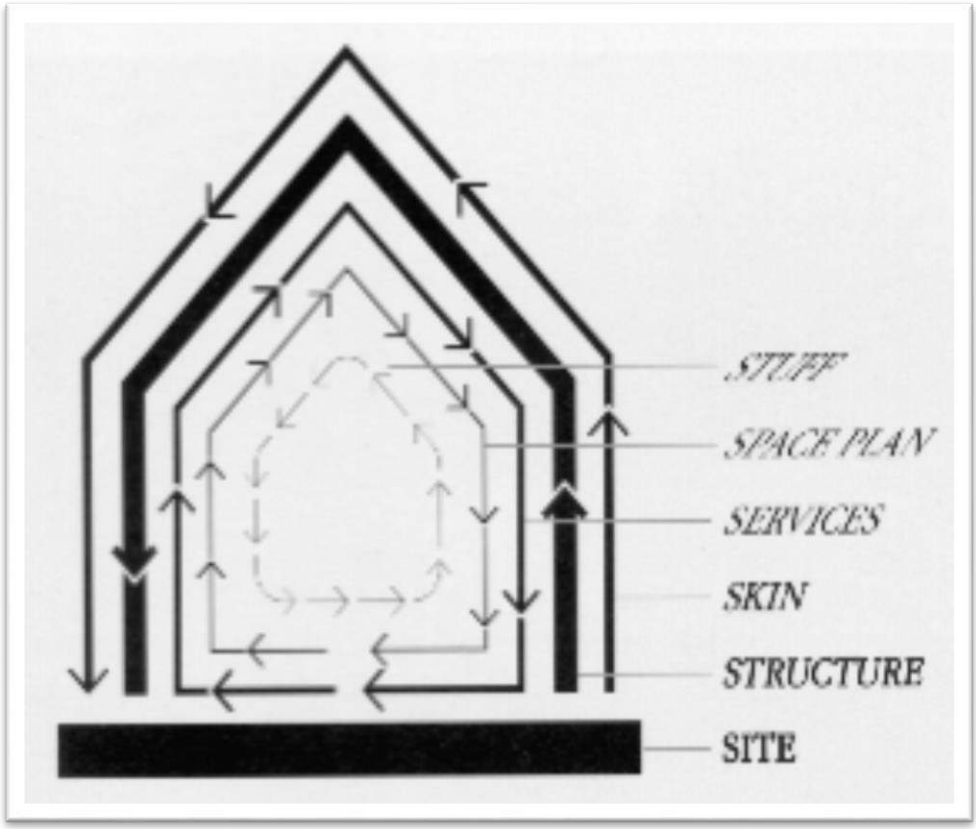


Stuart Brand's Shearing Layers

- Buildings components evolve at different timescales
- Layers: Each layer has its own value, and speed of change (pace)
- Buildings adapt because faster layers (services) are not obstructed by slower ones (structure)

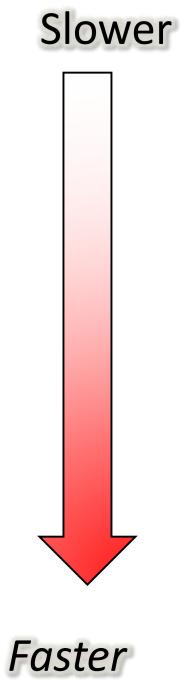
  —Stuart Brand,

  How Buildings Learn



Yoder and Foote’s Software Shearing Layers

“Factor your system so that artifacts that change at similar rates are together.”—Foote & Yoder, Big Ball of Mud Pattern

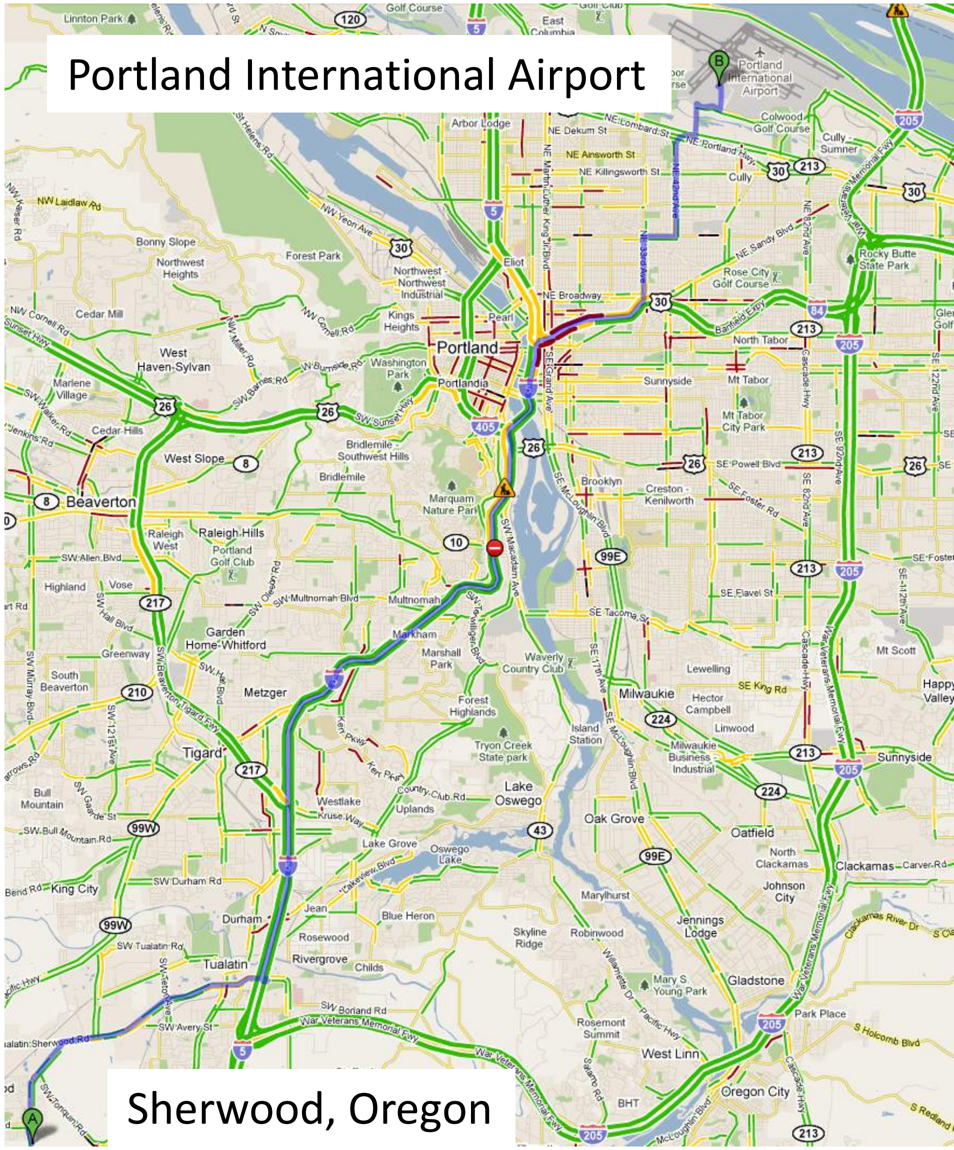


Layers

- The platform
- Infrastructure
- Data schema
- Standard frameworks and components
- Abstract classes and interfaces
- Classes
- Code
- Data

Product Roadmaps As Guides

- Where you expect to go
- What features and when? Relative time when feature is needed
- Influence architecture work and efforts

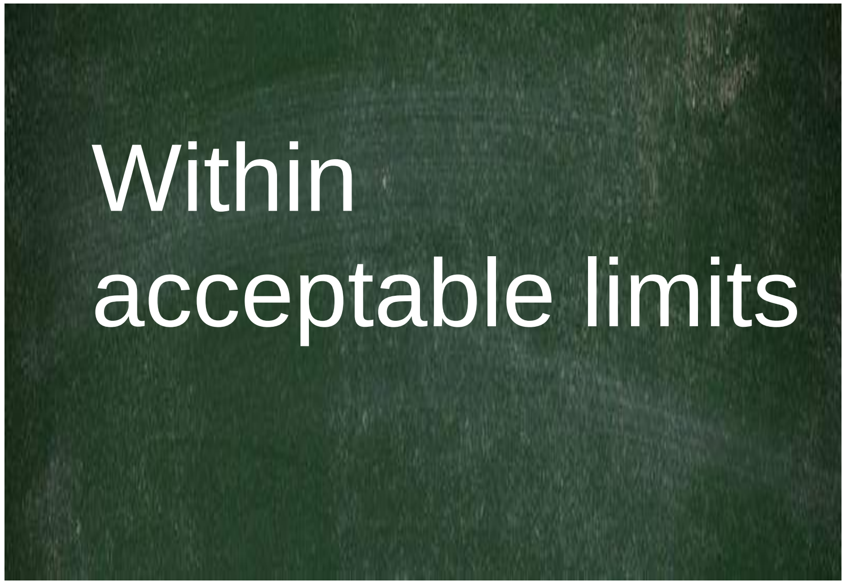


Product Landing Zones

- A range of acceptable values for important system qualities
  - ***Minimal***: OK, we can live with that
  - ***Target***: Realistic goal, what we are aiming for
  - ***Outstanding***: This would be great, if everything goes well

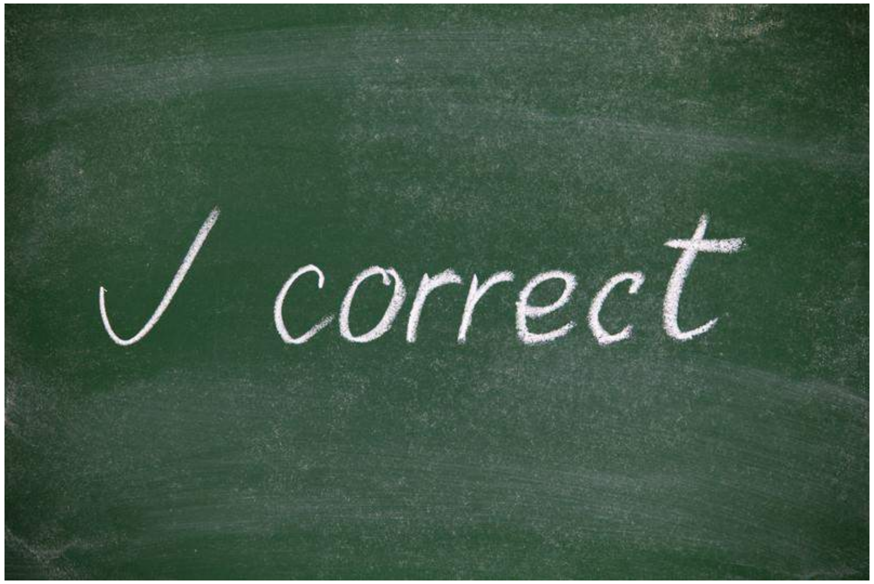
Good Landing Zone Criteria

- Define **acceptable range of values** for some characteristic or system quality (performance, usability, reliability, etc.)
  - – # transactions, average latency of a transaction under load, click through rate, up time….
- Broader in scope than an acceptance criteria
- SMART
    - Specific
    - Measurable
    - Achievable (minimum value)
    - Relevant
    - Timely



Good Acceptance Criteria

- Focused on a single thing (a rule or step of a process)
- A specification of what should happen/what must be true written in the language of the domain
- SMART
    - Specific
    - Measurable
    - Achievable
    - Relevant
    - Timely



What’s Different?

Acceptance Criteria

Free 2-day shipping is offered to Amazon Prime customers for all items in an order that are sold directly by Amazon

If an Amazon prime customer wants faster shipping, they pay standard shipping fees.

Automated tests can be written (fairly easily)

Landing Zone Criteria

Selection of shipping options should be completed with 99% customer accuracy

Test, but usually in production or staging environment

May require instrumenting “hooks” and making several measurements that are aggregated/interpreted

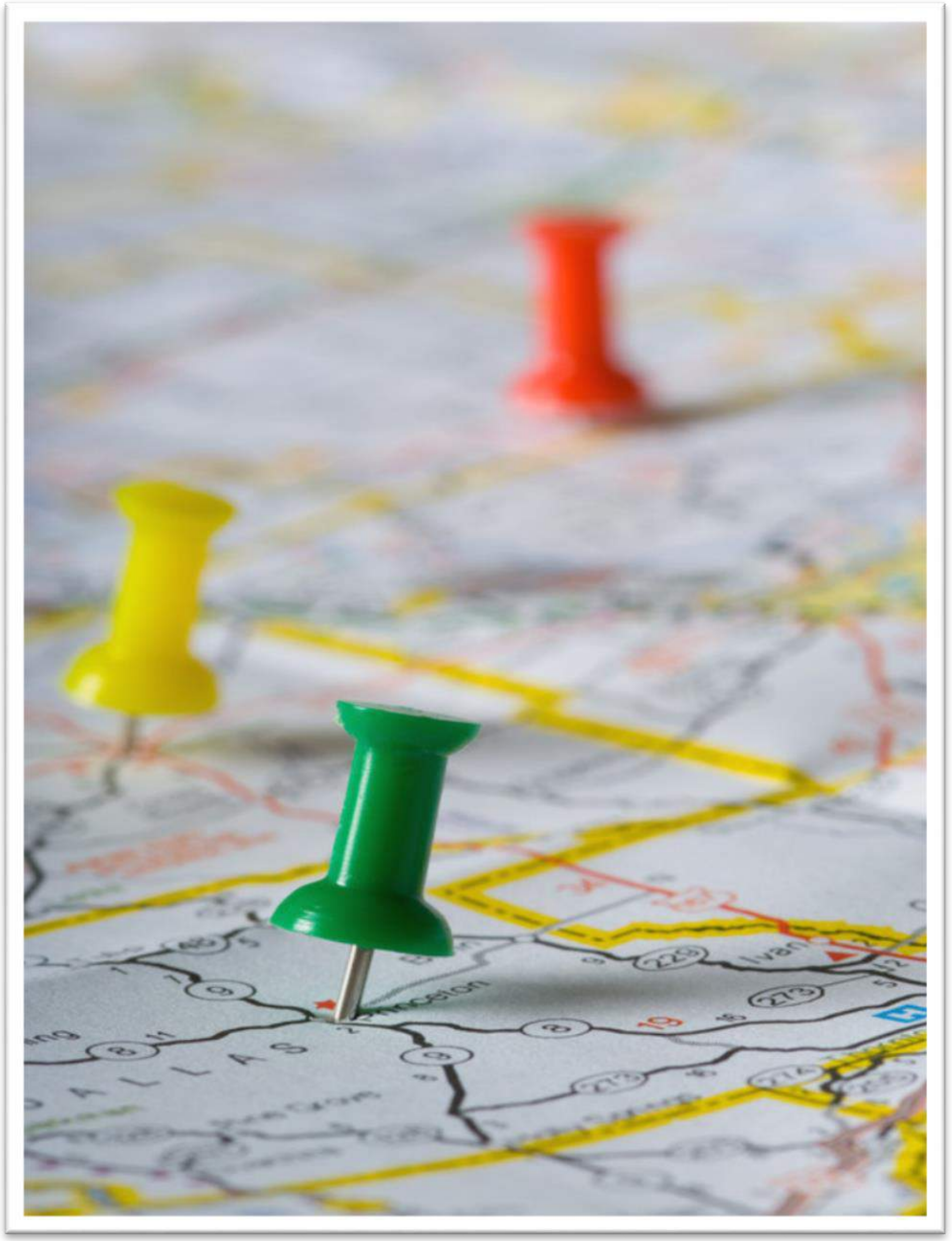
How Architects Use Landing Zones

- Create them with Product Owners and other Stakeholders
- Identify high architecture risk items
- Establish/verify target values
- Explain architecture tradeoffs and costs
- Monitor architecture health

Photo by e.r.w.i.n. Used with attribution
http://www.flickr.com/photos/eherrera/5104896694/

Landing Zones on Agile Projects

- Helps make sense of the bigger picture:
  - What happens when one attribute edges below minimum?
  - When will targets be achieved?
  - What do we need to do architecturally to achieve targets?



		Minimum	Target	Outstanding
Performance	Throughput (loan payment txns per day)	50,000	70,000	90,000
	Average loan payment txn time	2 seconds	1 second	< 1 second
Data Quality	Intersystem data consistency between x, y, z systems (per cent critical data attributes consistent)	95%	97%	97%
	ETL data accuracy for claims data	97%	99%	>99%

Managing Landing Zones

Too many criteria and you lose track of what's important

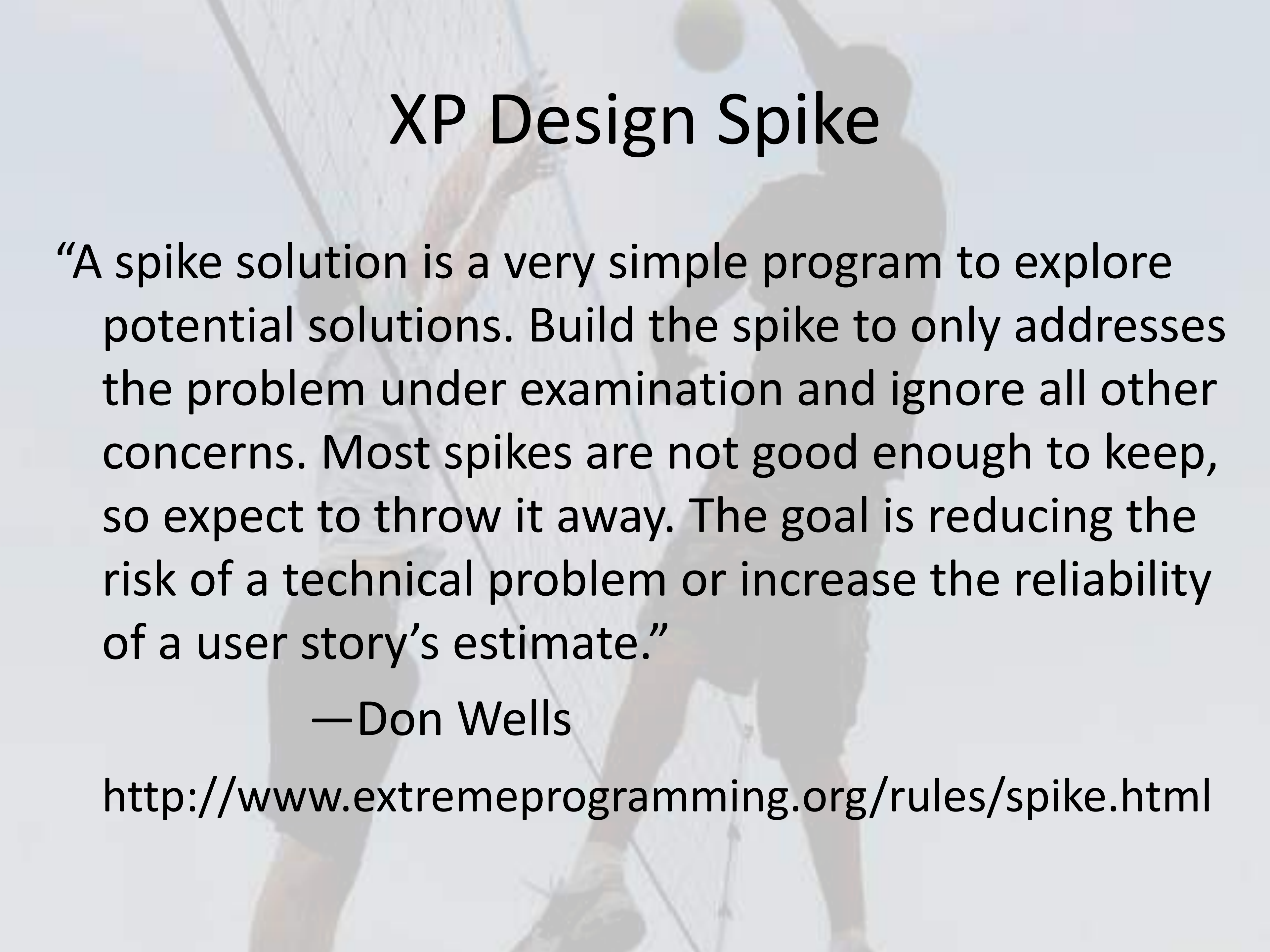
Define a core set, organize and group

Break down aggregate targets into measurable architecture-specific values

Be agile! Re-calibrate values as you implement more functionality

Architecture Spikes

- Bounded
- Explore potential solutions for achieving landing zone targets
- Not as tactical as an XP Design Spike
- Try out radical changes before committing to them



XP Design Spike

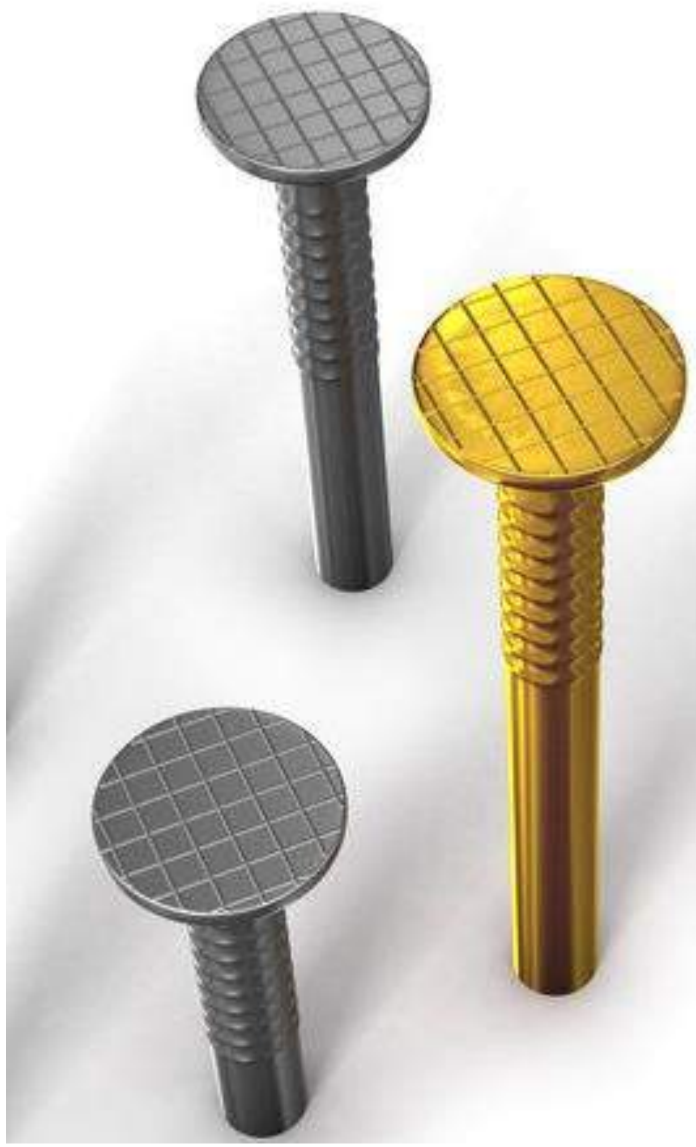
“A spike solution is a very simple program to explore potential solutions. Build the spike to only addresses the problem under examination and ignore all other concerns. Most spikes are not good enough to keep, so expect to throw it away. The goal is reducing the risk of a technical problem or increase the reliability of a user story’s estimate.”

—Don Wells

http://www.extremeprogramming.org/rules/spike.html

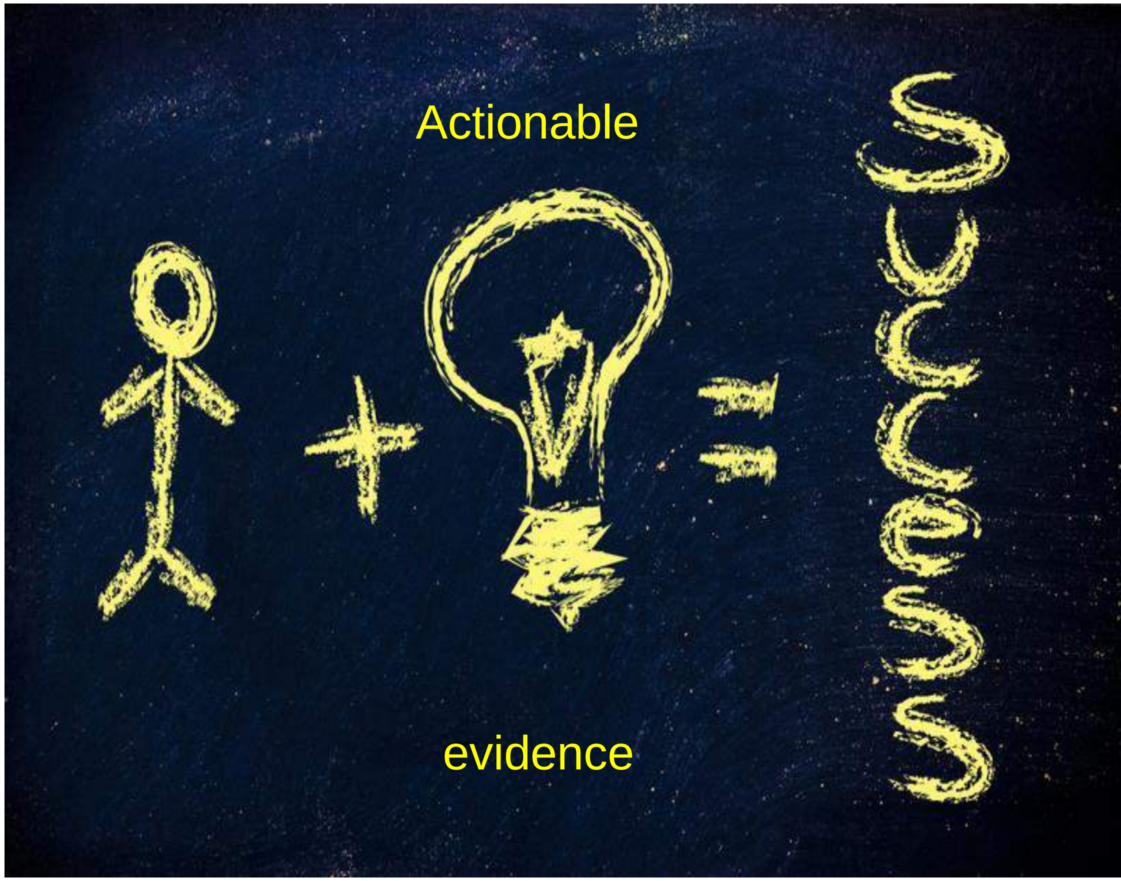
What You Do In an Architecture Spike

- prototyping
- design noodling
- looking outside
- experimenting
- modeling
- proving ideas



Criteria For an Architecture Spike: *Actionable Results*

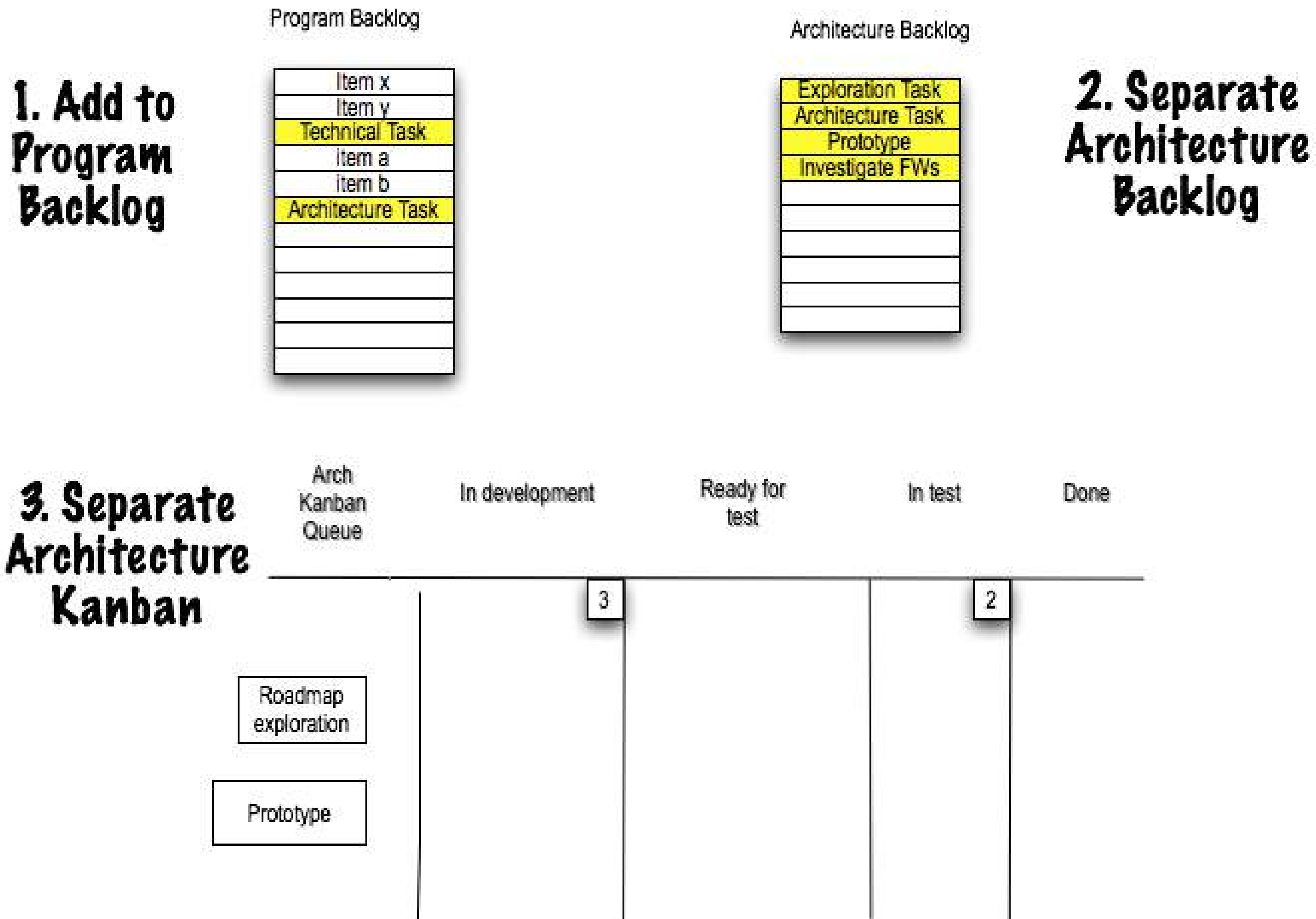
- Buys information
  - Feasibility
  - Reasonable approach
  - Alternatives
- Feeds into planning
  - Adjusts the release roadmap
  - Recalibrates landing zone
  - Drives new development and design

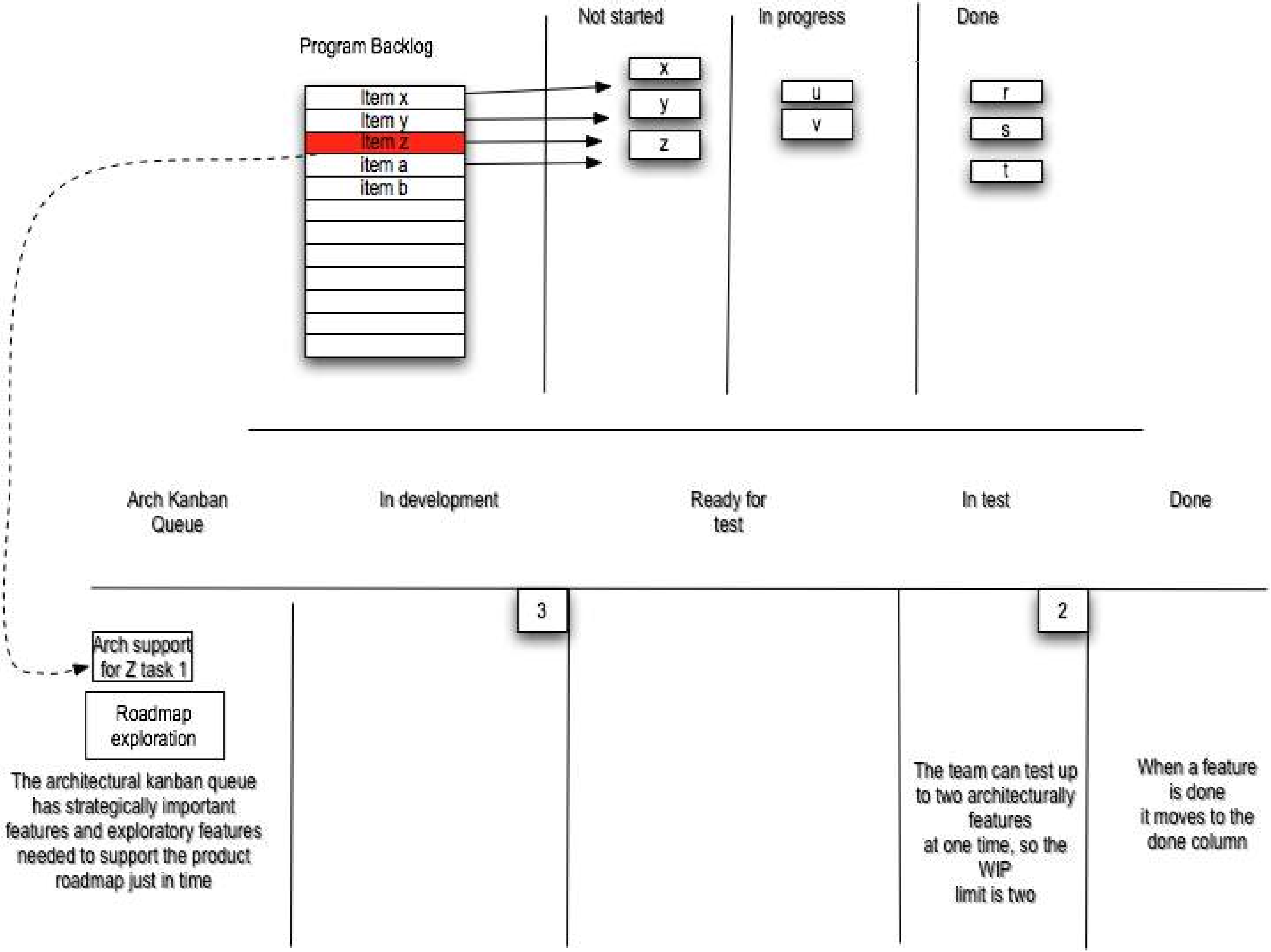


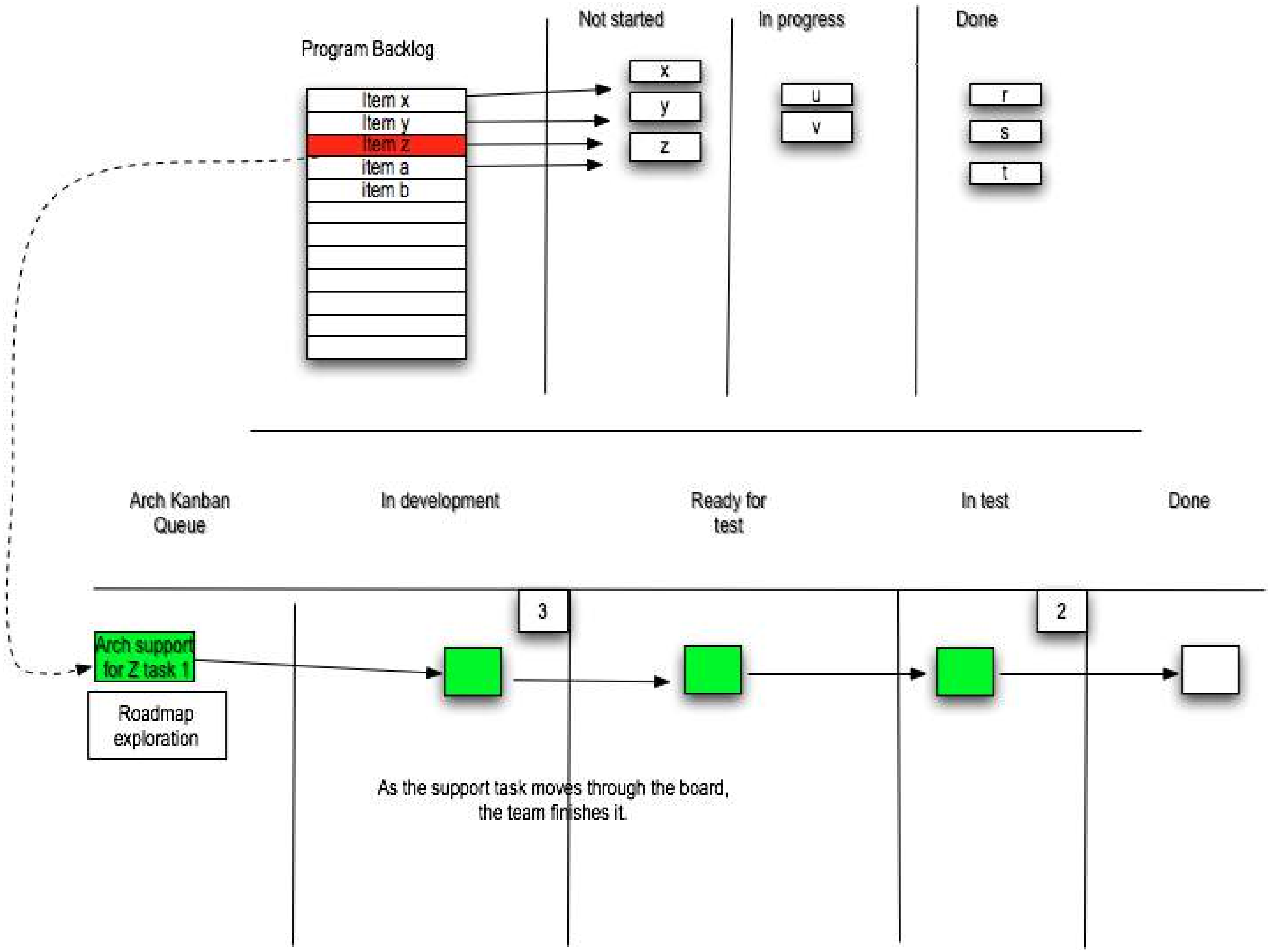
Architecture Spike Best Practices

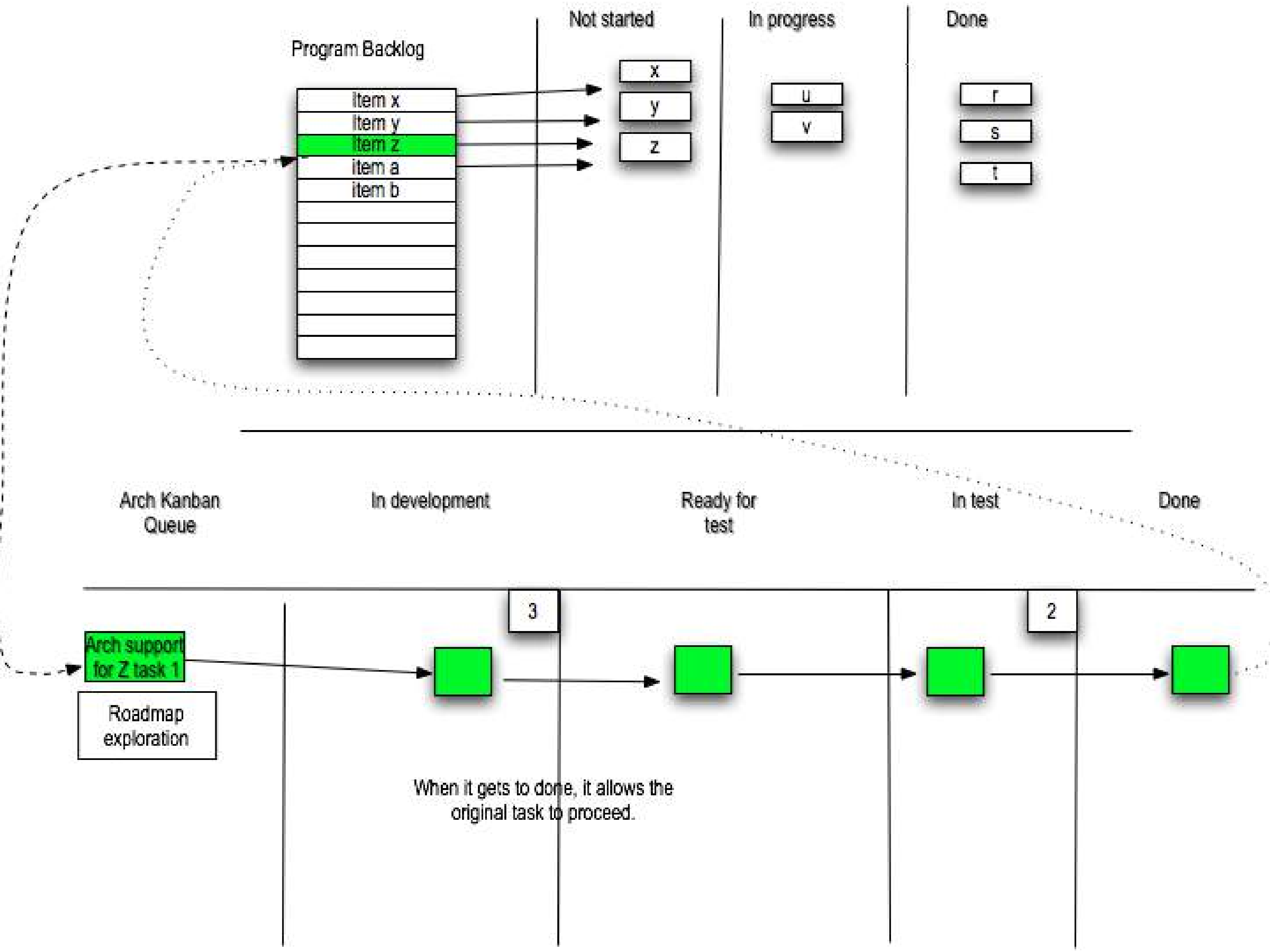
- Small, smart, goal-oriented teams
  - avoid us vs. them mentality
- Evidence-based answers
  - working prototypes
  - existing similar things
- Time-boxed
  - Limited scope and duration (2-6 weeks)
- **Failure is an option**
  - permit answers that may shift goals

3 Ways To Manage Architectural Tasks




Program Backlog
Not started
In progress
Done
Item x
Item y
Item z
item a
item b
x
y
z
u
v
r
s
t
Arch Kanban Queue
In development
Ready for test
In test
Done
3
2
Arch support for Z task 1
Roadmap exploration
The architectural kanban queue has strategically important features and exploratory features needed to support the product roadmap just in time
The team can test up to two architecturally features at one time, so the WIP limit is two
When a feature is done it moves to the done column


Program Backlog
Not started
In progress
Done
Item x
Item y
Item z
item a
item b
x
y
z
u
v
r
s
t
Arch Kanban Queue
In development
Ready for test
In test
Done
3
2
Arch support for Z task 1
Roadmap exploration
As the support task moves through the board, the team finishes it.



When it gets to done, it allows the original task to proceed.

What Can Go On An Architecture Backlog?

Architecturally meaty feature

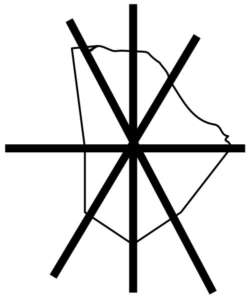
Design spike related task

Architecture investigation

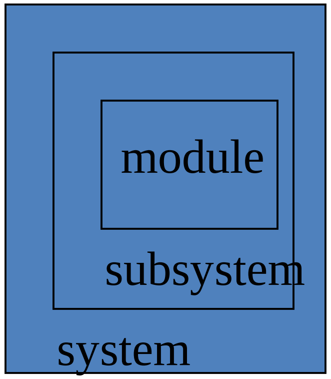
Prototype

Framework development

Roadmap exploration

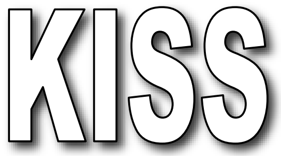


balance

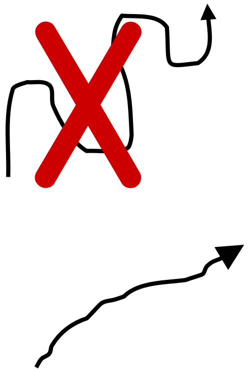


system structure
design approach

architecture
views,
explanations,
sketches



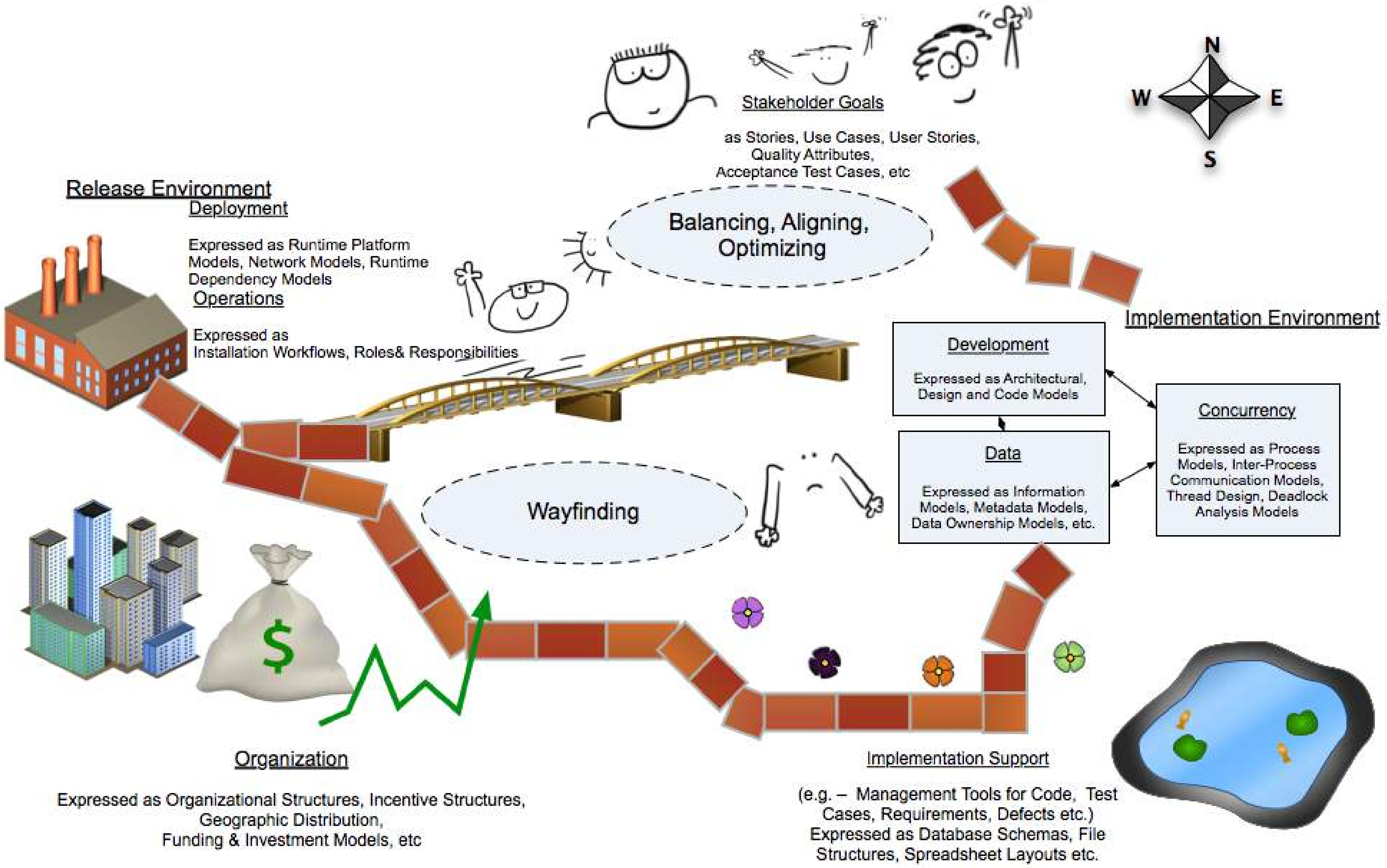
elegance



system
integrity and
sustainability

WHAT DO AGILE ARCHITECTS DO?

The Agile Architecture Landscape


Stakeholder Goals
as Stories, Use Cases, User Stories, Quality Attributes, Acceptance Test Cases, etc
N
W
E
S
Release Environment
Deployment
Expressed as Runtime Platform Models, Network Models, Runtime Dependency Models
Operations
Expressed as Installation Workflows, Roles& Responsibilities
Balancing, Aligning, Optimizing
Implementation Environment
Development
Expressed as Architectural, Design and Code Models
Concurrency
Expressed as Process Models, Inter-Process Communication Models, Thread Design, Deadlock Analysis Models
Data
Expressed as Information Models, Metadata Models, Data Ownership Models, etc.
Wayfinding
$
Organization
Expressed as Organizational Structures, Incentive Structures, Geographic Distribution, Funding & Investment Models, etc
Implementation Support
(e.g. – Management Tools for Code, Test Cases, Requirements, Defects etc.) Expressed as Database Schemas, File Structures, Spreadsheet Layouts etc.

Differences Between Agile and Traditional Architecture

Traditional

- Big picture thinking
- Produces Models and blue prints
- Not so hands-on
- Focused on compliance

Agile

- Balances big picture & details
- Produce what's needed to make informed decisions
- Hands-on
- Focused on sustainability

Models

“Big M”	vs.	“little m”
• Lots of time invested		• Not a lot of time invested
• Intended to last		• Intended to communicate
• “Definitive”		• Often discarded
• Usually formal		• Can be formal or informal
• May not be widely used or understood		• Made to be viewed

Agile architects create models as needed

CRC Cards: A “little m” model

The First CRC Cards

“A Laboratory For Teaching Object-Oriented Thinking,”
Kent Beck, Apple Computer, Inc., Ward Cunningham, Wyatt Software Services, Inc.
OOPSLA 89

Model	
Maintain problem related info	
Broadcast change notification	

View	
Render the model	Model
Transform coordinates	Controller

Controller	
Interpret user input	Model
Distribute control	View

Example: Database “Responsibilities”

Supports interactive web and self service applications

Provides storage for:

- Transactions that will affect systems of record
- Staging information closer to the user to support high performance access
- Data required by end users that comes from systems of record that do not have 24 x 7 availability

Supports the event driven and service integration architecture.

Provides storage for:

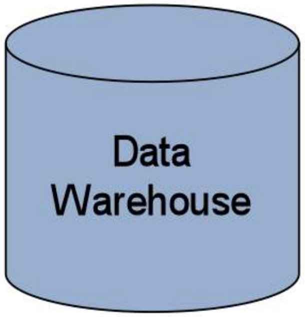
- transformation and enrichment services
- long running transactions.
- audit and performance metrics
- messages that need replayed in case of an unexpected failure
- error handling

Repository for those business entities that are shared across systems of record

- Customer is an example on such an entity
- Is responsible for managing the synchronizing those entities across systems
- Fundamentally a store for business identity management

Repository for business data and transactions

- Based on business processes
- Considered the single source of the truth as it relates to a given entity
- A given entity should have one and only one system of record



Supports capturing and storing data to support reporting and business analytics

Provides Storage for

- Time variant/non volatile data sourced from systems of record
- Historical record of transactional data
- Archival data for those systems of records not capable to support historical tracking of data

Values Important to Agile Architects

- Balance
- Testable architectural qualities
- Hands-on
  - programming, designing, reading code, building things…

Agile Values Drive Architectural Practices

- Sustainable development
- Responsible moments
- Evidence-based decisions
- Attention to detail

Indicators You've Paid Enough Attention to Architecture

- Defects localized
- Stable interfaces
- Consistency
- Performant
- New functionality doesn't "break" existing architecture
- Few areas developers avoid because they are too unpleasant to work in

Thank you

-Rebecca
rebecca@wirfs-brock.com
Twitter: @rebeccawb
Additional Resources:
2 day Agile Architecture Workshop
Being Agile About System Qualities Workshop
The Responsible Designer Blog:
www.wirfs-brock.com/blog