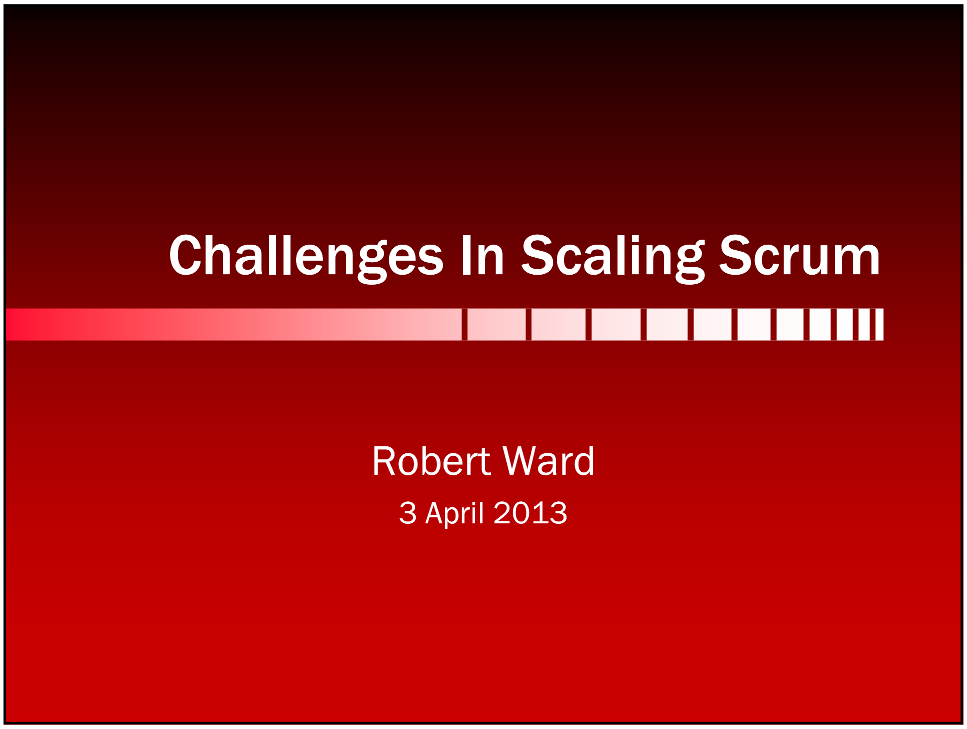
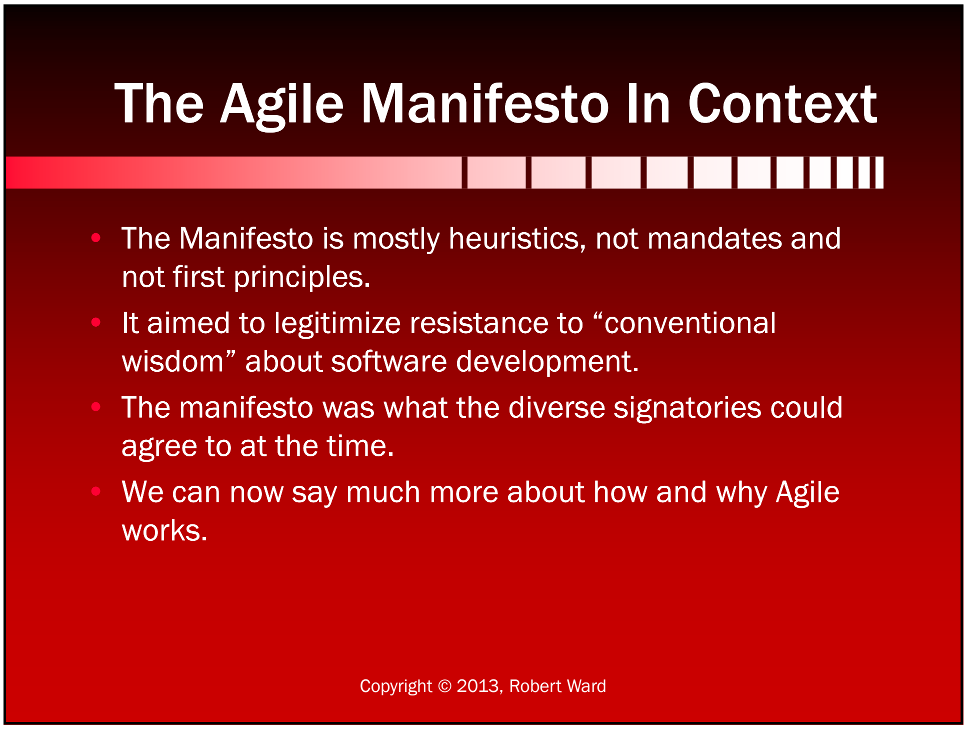


Challenges In Scaling Scrum

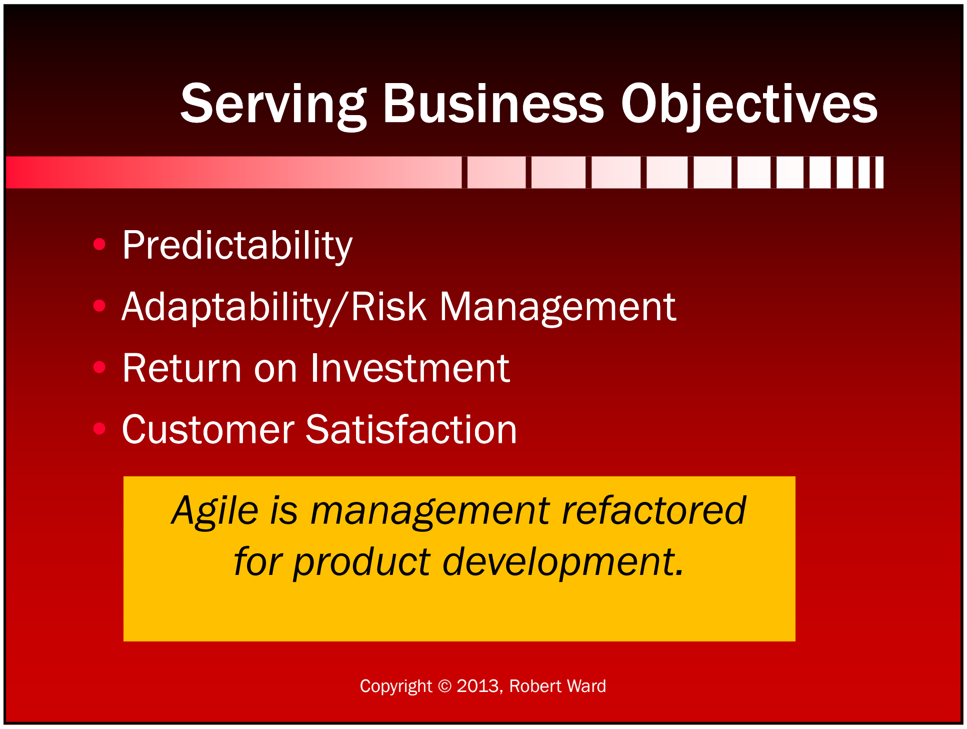
Robert Ward

3 April 2013



The Agile Manifesto In Context

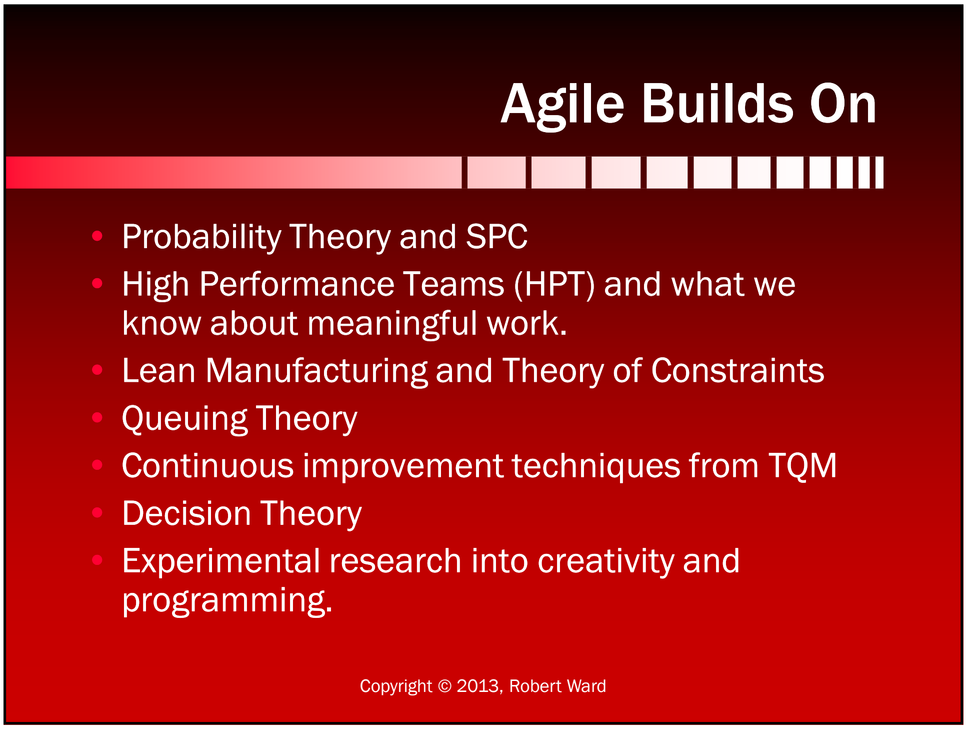
- The Manifesto is mostly heuristics, not mandates and not first principles.
- It aimed to legitimize resistance to “conventional wisdom” about software development.
- The manifesto was what the diverse signatories could agree to at the time.
- We can now say much more about how and why Agile works.



Serving Business Objectives

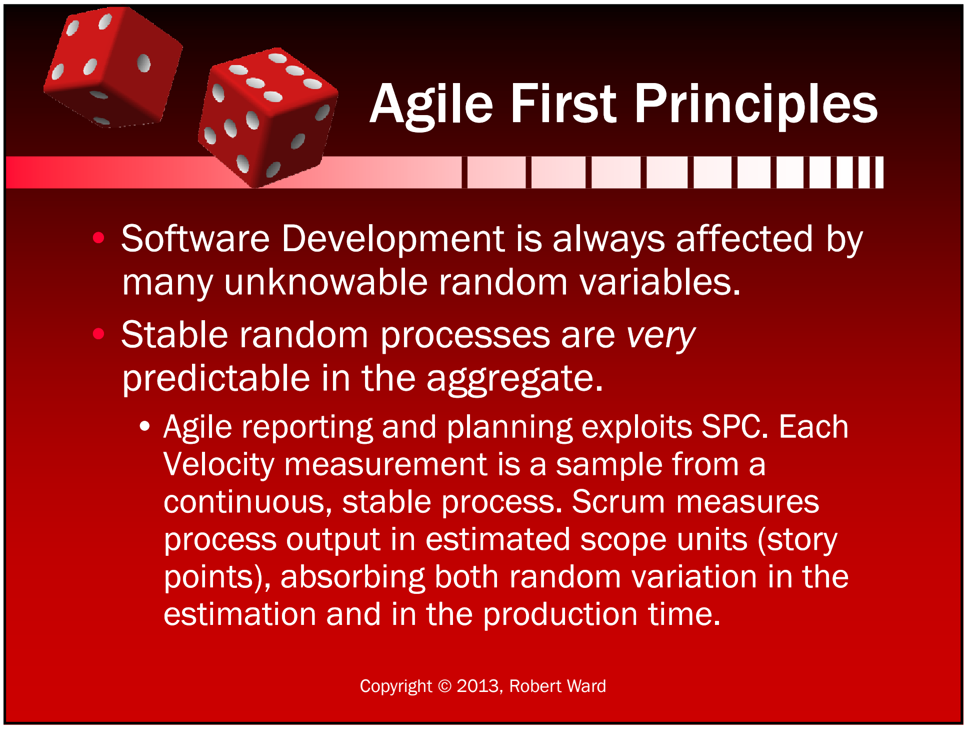
- Predictability
- Adaptability/Risk Management
- Return on Investment
- Customer Satisfaction

Agile is management refactored for product development.



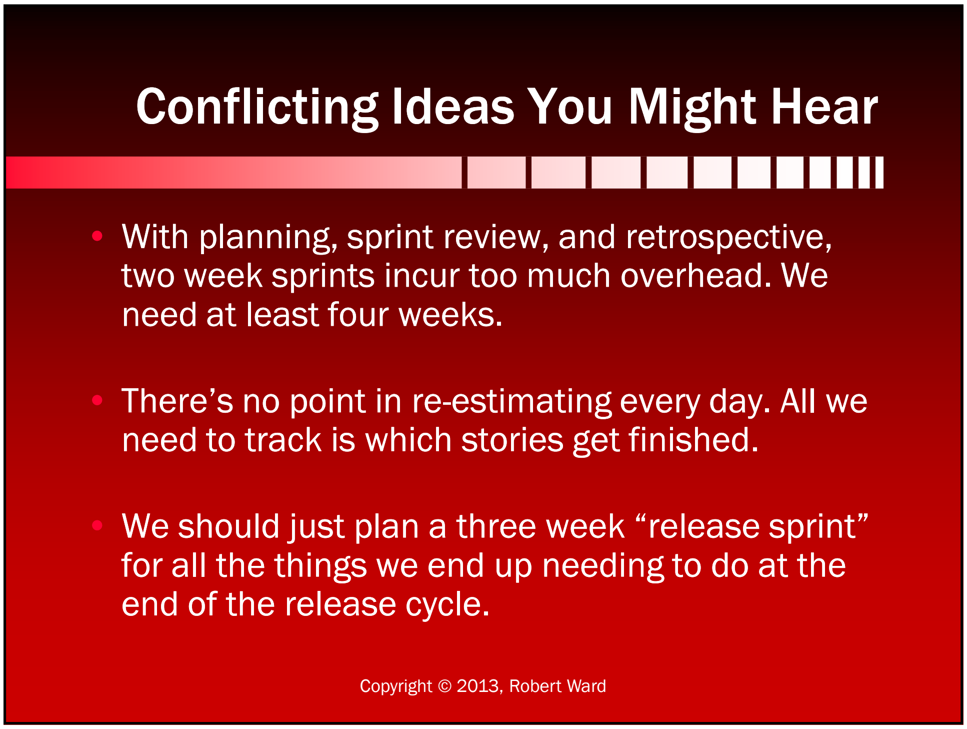
Agile Builds On

- Probability Theory and SPC
- High Performance Teams (HPT) and what we know about meaningful work.
- Lean Manufacturing and Theory of Constraints
- Queuing Theory
- Continuous improvement techniques from TQM
- Decision Theory
- Experimental research into creativity and programming.



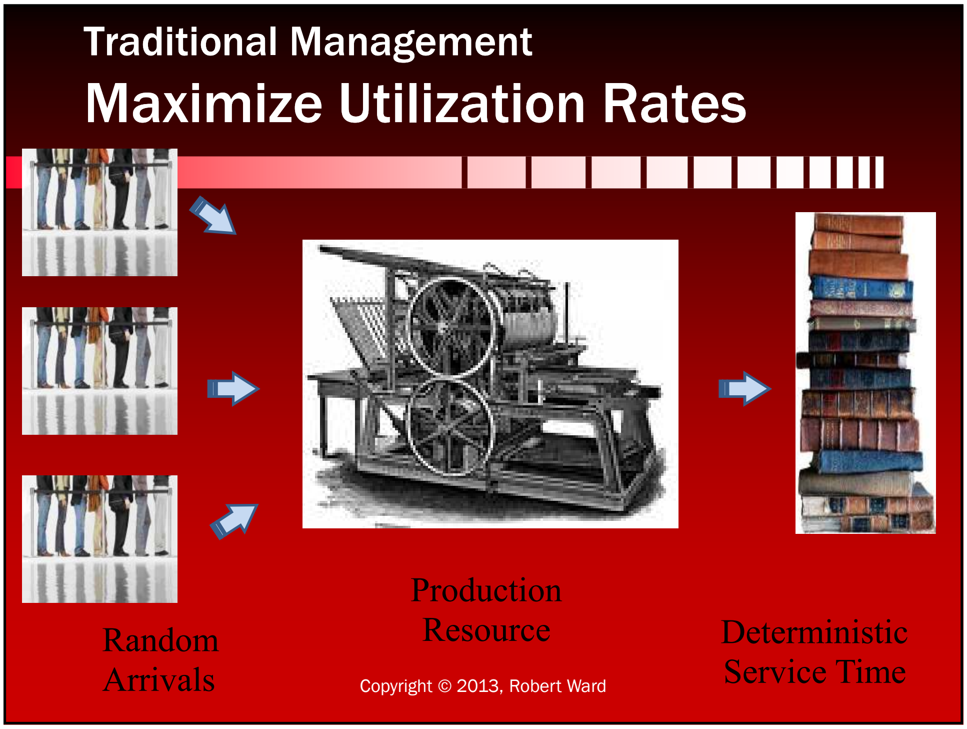
Agile First Principles

- Software Development is always affected by many unknowable random variables.
- Stable random processes are *very* predictable in the aggregate.
  - Agile reporting and planning exploits SPC. Each Velocity measurement is a sample from a continuous, stable process. Scrum measures process output in estimated scope units (story points), absorbing both random variation in the estimation and in the production time.



Conflicting Ideas You Might Hear

- With planning, sprint review, and retrospective, two week sprints incur too much overhead. We need at least four weeks.
- There's no point in re-estimating every day. All we need to track is which stories get finished.
- We should just plan a three week "release sprint" for all the things we end up needing to do at the end of the release cycle.



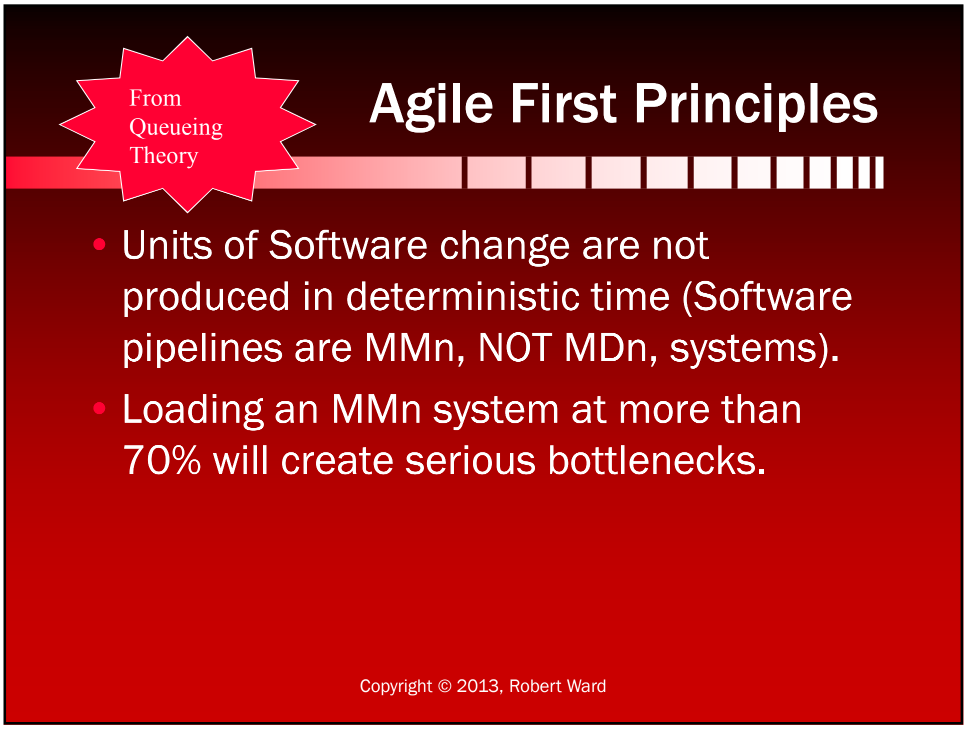
Traditional Management

Maximize Utilization Rates

Random Arrivals

Production Resource

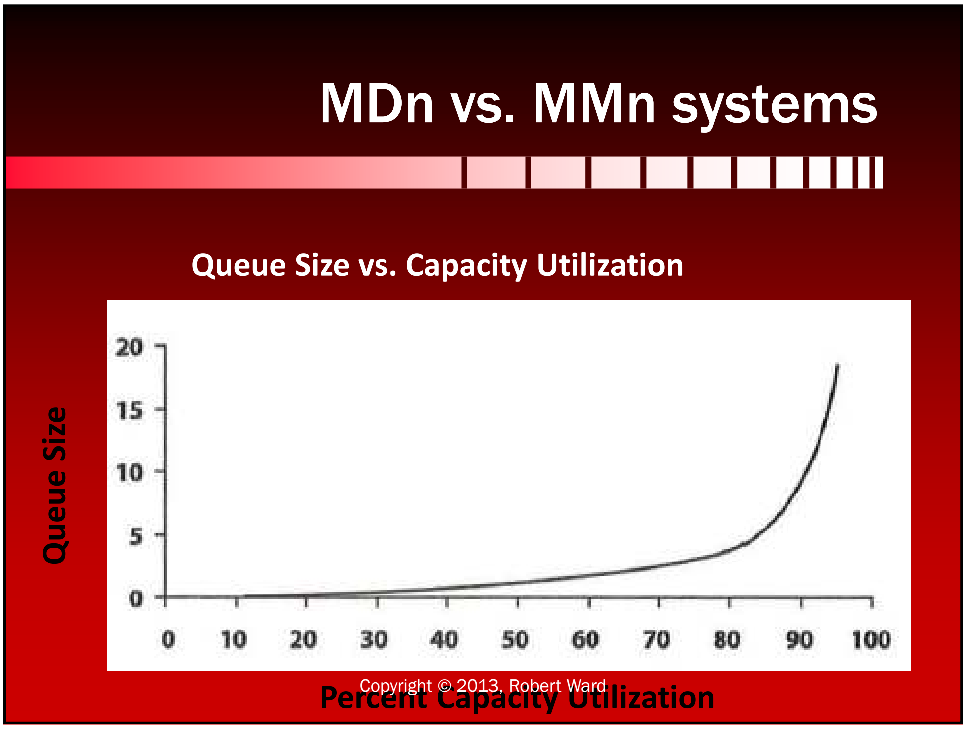
Deterministic Service Time



From Queueing Theory

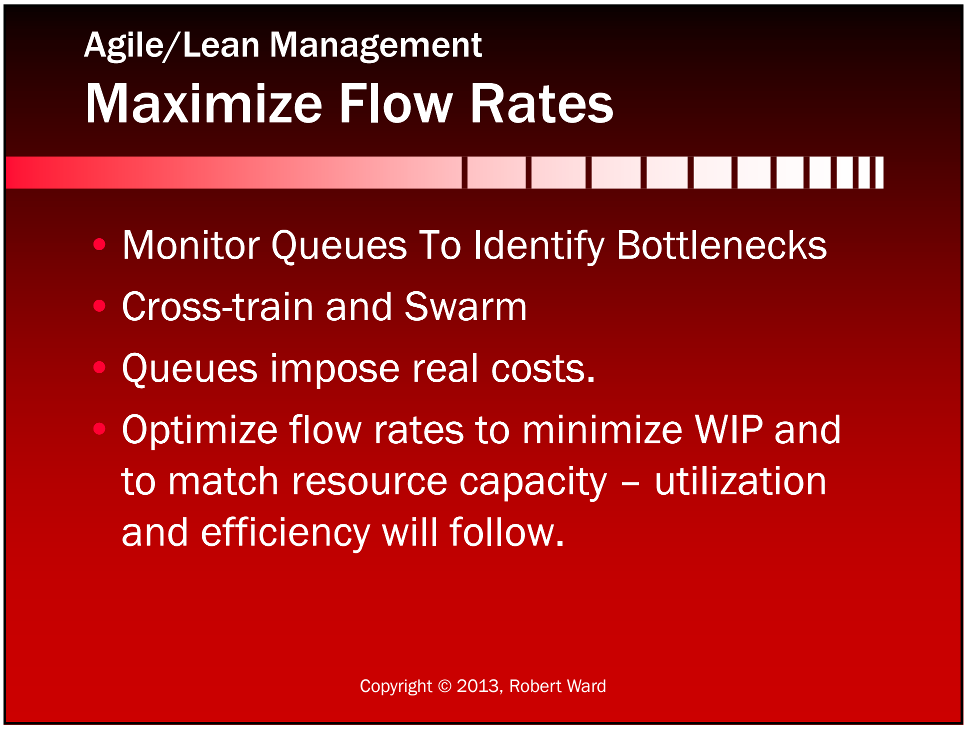
Agile First Principles

- Units of Software change are not produced in deterministic time (Software pipelines are MMn, NOT MDn, systems).
- Loading an MMn system at more than 70% will create serious bottlenecks.



MDn vs. MMn systems

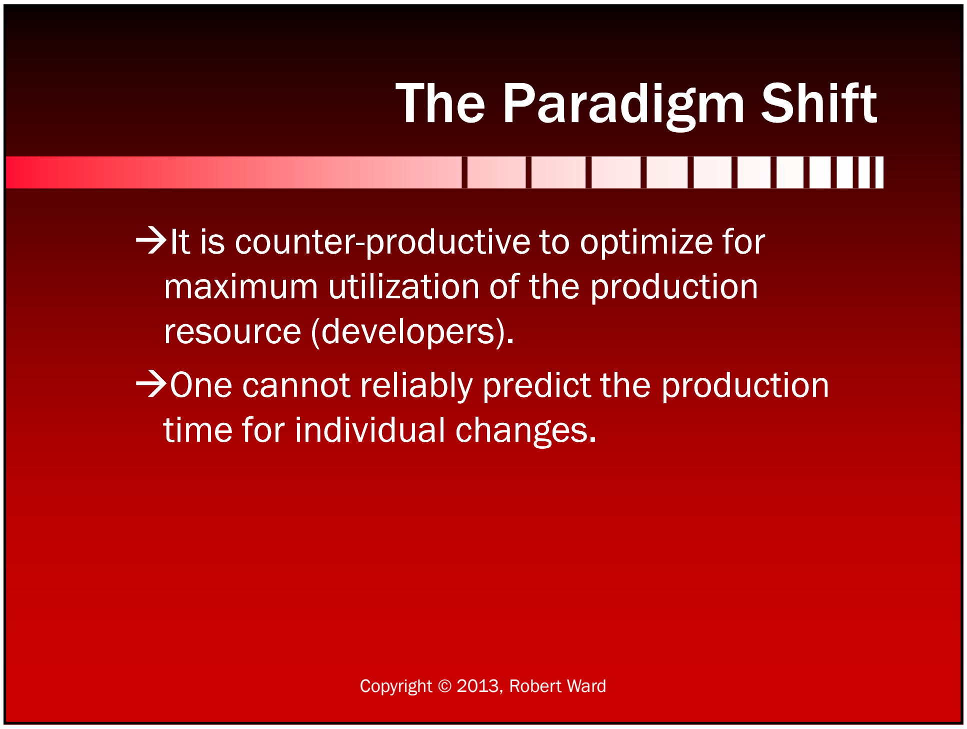
Queue Size vs. Capacity Utilization



Agile/Lean Management

Maximize Flow Rates

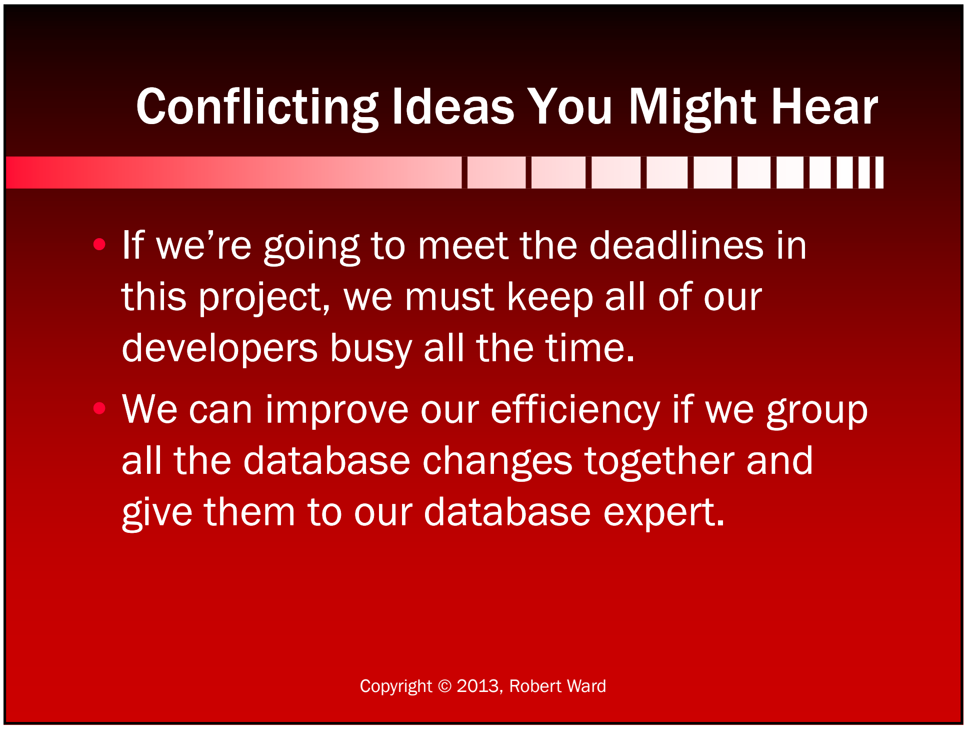
- Monitor Queues To Identify Bottlenecks
- Cross-train and Swarm
- Queues impose real costs.
- Optimize flow rates to minimize WIP and to match resource capacity – utilization and efficiency will follow.



The Paradigm Shift

→It is counter-productive to optimize for maximum utilization of the production resource (developers).

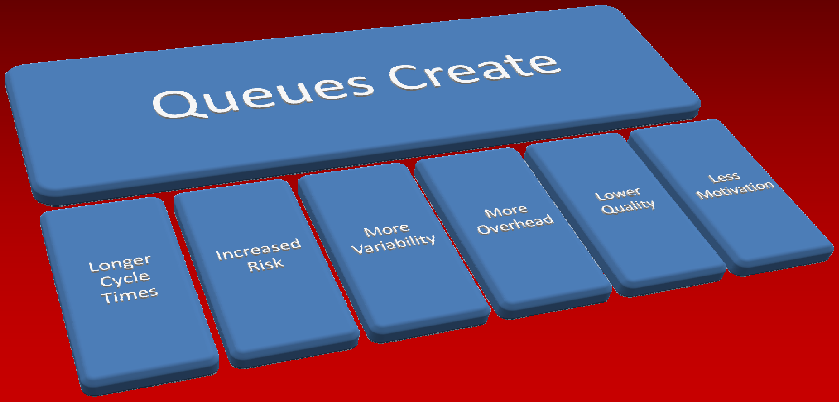
→One cannot reliably predict the production time for individual changes.



Conflicting Ideas You Might Hear

- If we’re going to meet the deadlines in this project, we must keep all of our developers busy all the time.
- We can improve our efficiency if we group all the database changes together and give them to our database expert.

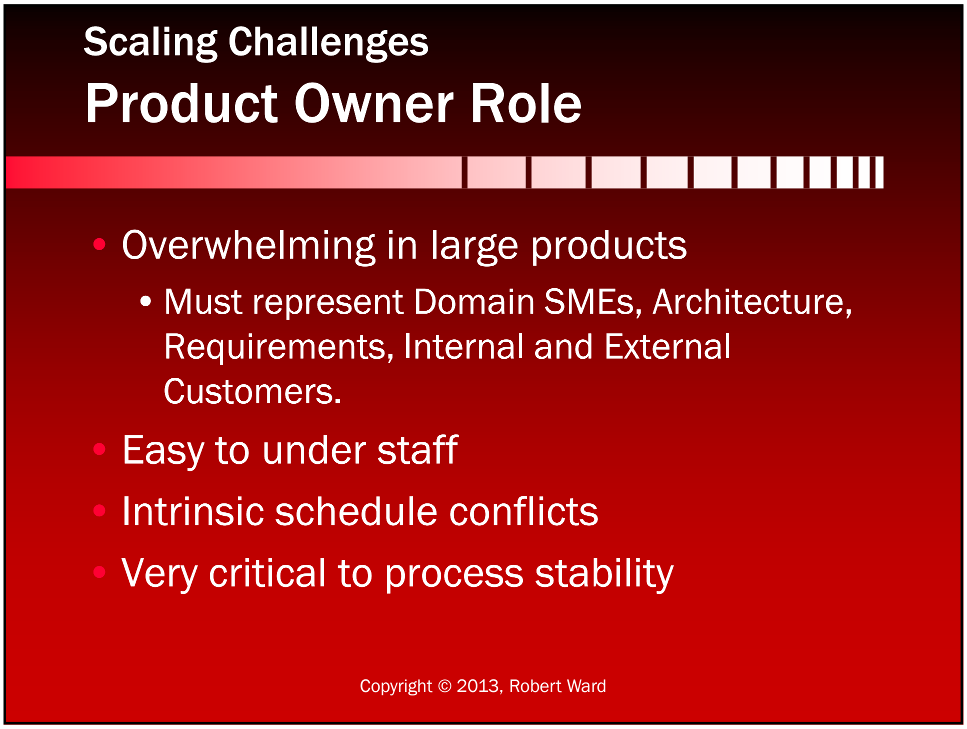
The Cost of Queues



More From Queueing Theory

Agile First Principles

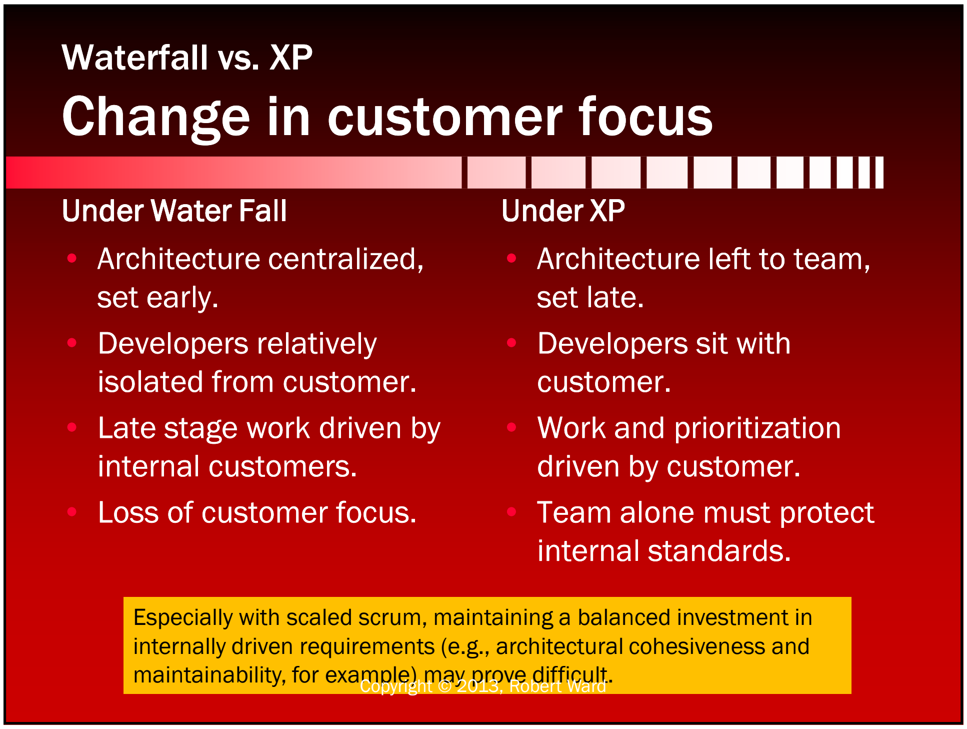
- Minimum average service time comes from single queue, multiple “all purpose” servers.
- Provided transaction costs are kept low enough, reducing packet size improves flow rates.



Scaling Challenges

Product Owner Role

- Overwhelming in large products
  - Must represent Domain SMEs, Architecture, Requirements, Internal and External Customers.
- Easy to under staff
- Intrinsic schedule conflicts
- Very critical to process stability



Waterfall vs. XP

Change in customer focus

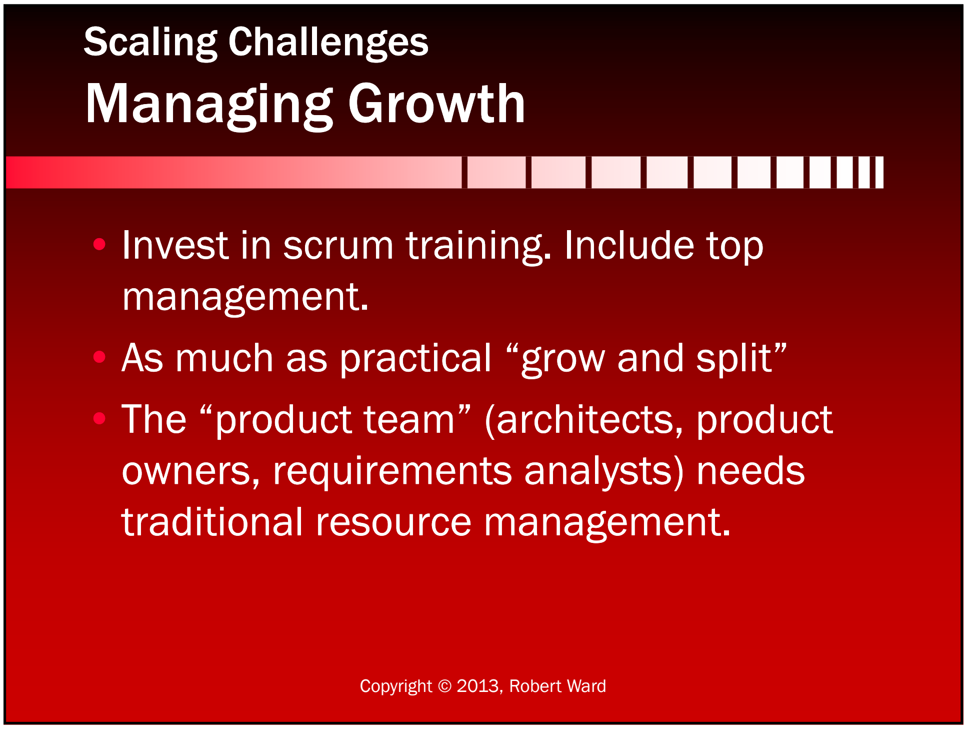
Under Water Fall

- Architecture centralized, set early.
- Developers relatively isolated from customer.
- Late stage work driven by internal customers.
- Loss of customer focus.

Under XP

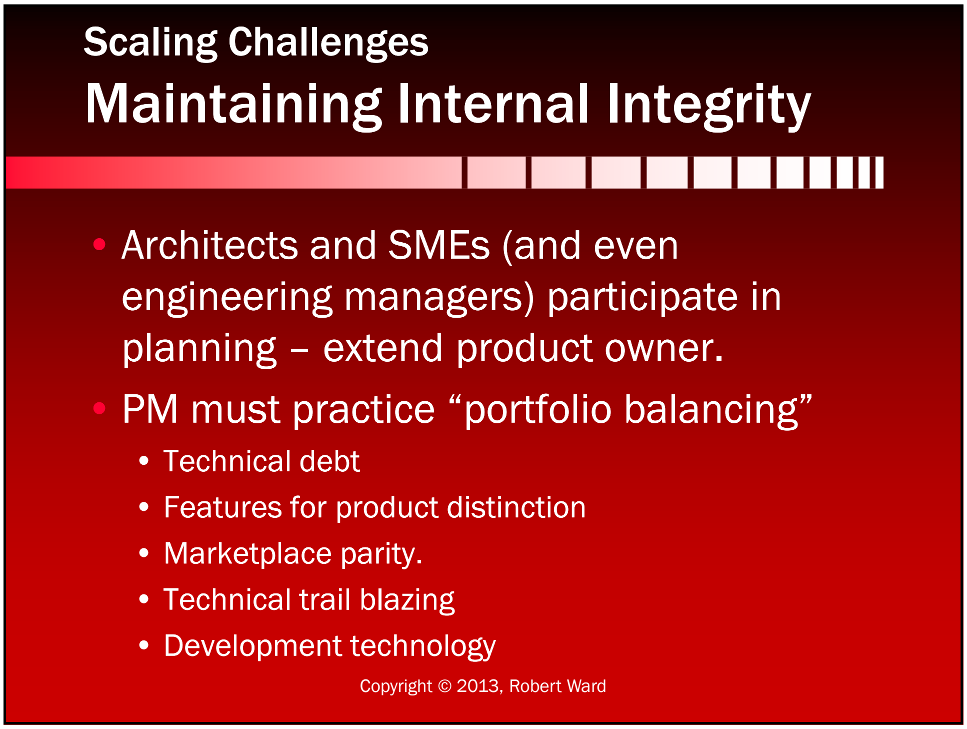
- Architecture left to team, set late.
- Developers sit with customer.
- Work and prioritization driven by customer.
- Team alone must protect internal standards.

Especially with scaled scrum, maintaining a balanced investment in internally driven requirements (e.g., architectural cohesiveness and maintainability, for example) may prove difficult.



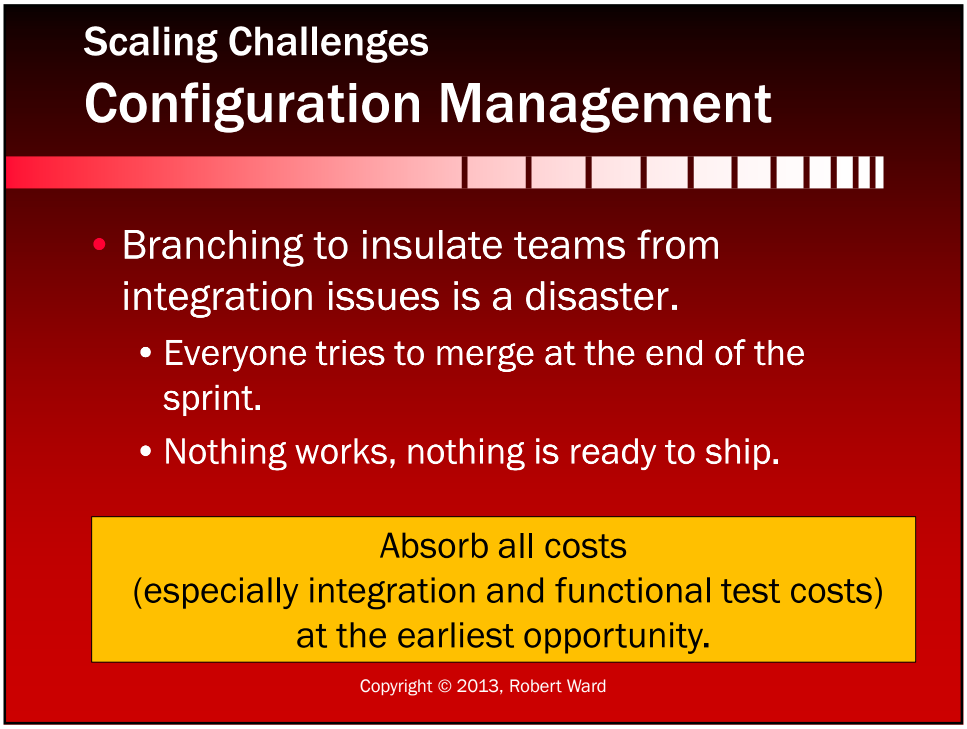
Scaling Challenges
Managing Growth

- Invest in scrum training. Include top management.
- As much as practical “grow and split”
- The “product team” (architects, product owners, requirements analysts) needs traditional resource management.



Scaling Challenges
Maintaining Internal Integrity

- Architects and SMEs (and even engineering managers) participate in planning – extend product owner.
- PM must practice “portfolio balancing”
  - Technical debt
  - Features for product distinction
  - Marketplace parity.
  - Technical trail blazing
  - Development technology

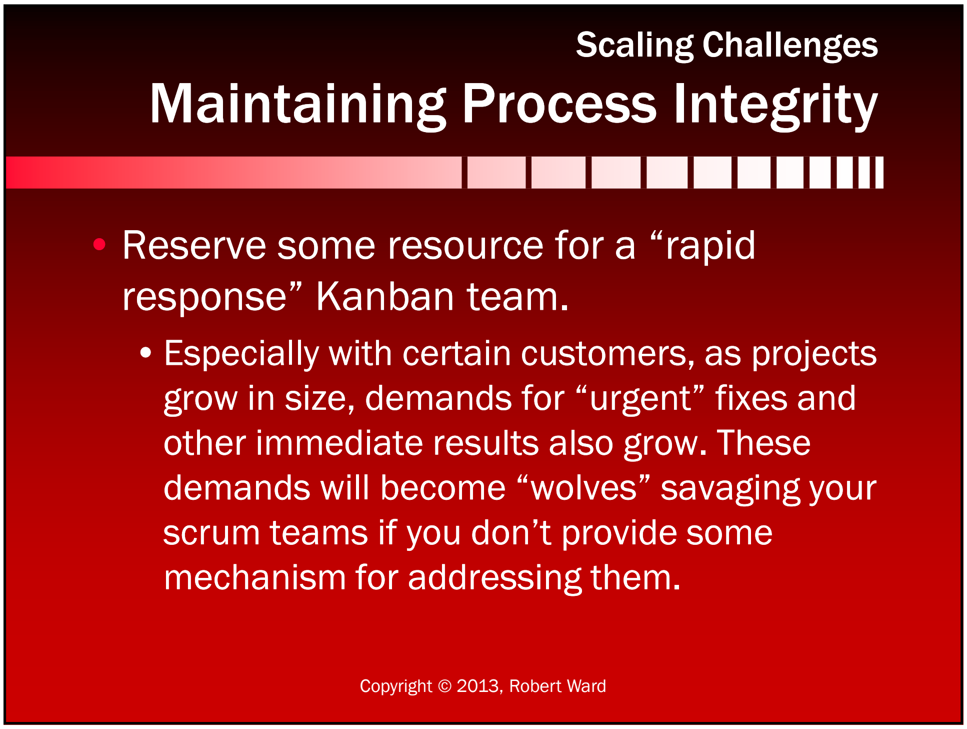


Scaling Challenges

Configuration Management

- Branching to insulate teams from integration issues is a disaster.
  - Everyone tries to merge at the end of the sprint.
  - Nothing works, nothing is ready to ship.

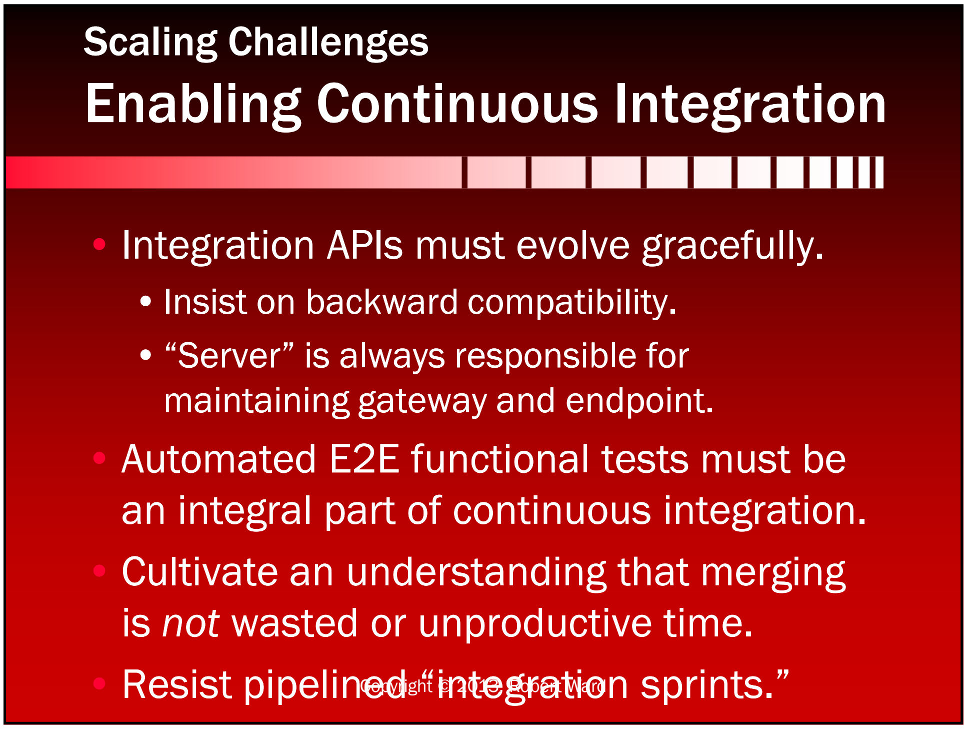
Absorb all costs
(especially integration and functional test costs)
at the earliest opportunity.



Scaling Challenges

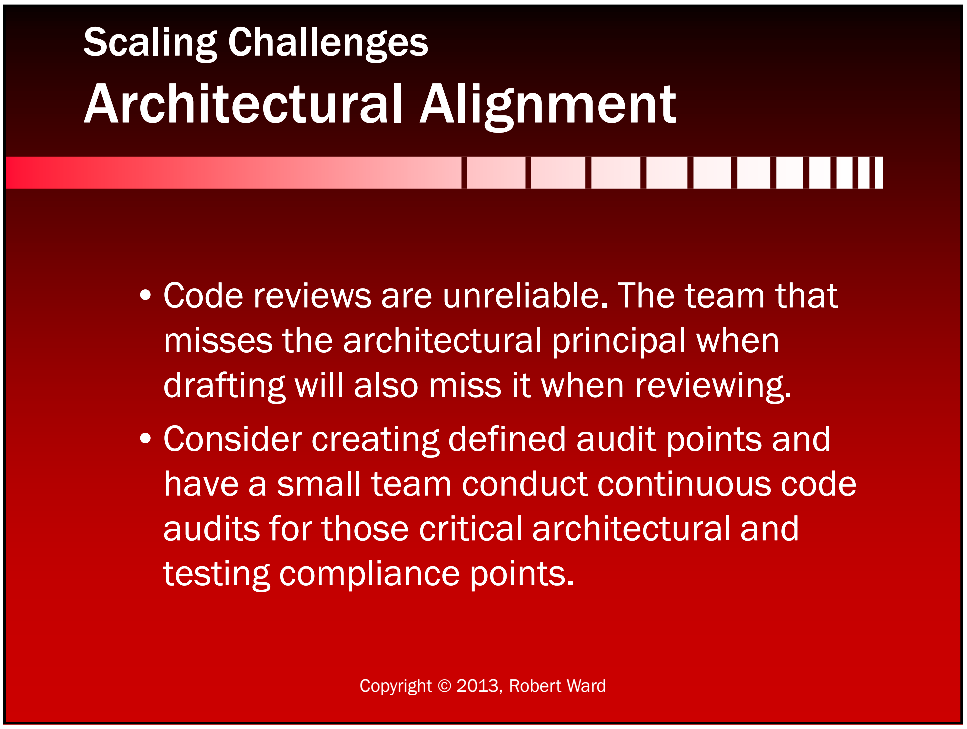
Maintaining Process Integrity

- Reserve some resource for a “rapid response” Kanban team.
  - Especially with certain customers, as projects grow in size, demands for “urgent” fixes and other immediate results also grow. These demands will become “wolves” savaging your scrum teams if you don’t provide some mechanism for addressing them.



Scaling Challenges
Enabling Continuous Integration

- Integration APIs must evolve gracefully.
  - Insist on backward compatibility.
  - “Server” is always responsible for maintaining gateway and endpoint.
- Automated E2E functional tests must be an integral part of continuous integration.
- Cultivate an understanding that merging is *not* wasted or unproductive time.
- Resist pipelined “integration sprints.”



Scaling Challenges
Architectural Alignment

- Code reviews are unreliable. The team that misses the architectural principal when drafting will also miss it when reviewing.
- Consider creating defined audit points and have a small team conduct continuous code audits for those critical architectural and testing compliance points.

Summary

- Scrum can significantly improve developer job satisfaction, but that is a secondary result. Business objectives are driving.
- Scrum works best when management understands the connections to SPC, Lean, HPT, Queuing theory, etc.
- Scrum is not just a "tweak" on traditional management, it is a major paradigm shift.

Resources

Deming, William Edwards. *Out of the Crisis*. MIT press, 2000.

Buchholz, Steve, Thomas Roth, and Kären M. Hess. *Creating the high performance team*. John Wiley & Sons Inc, 1987.

Reinertsen, D. G. *The Principles of Product Development Flow–Second Generation Lean Product Development.* Celeritas Publishing, 2009.

Carson, Shelley. *Your creative brain: Seven steps to maximize imagination, productivity, and innovation in your life*. Jossey-Bass, 2010.

Gladwell, Malcolm. *Outliers: The story of success*. ePenguin, 2008.

Martin, Robert C. *Clean code: a handbook of agile software craftsmanship*. Prentice Hall, 2008.

Cohn, Mike. *Succeeding with agile: software development using Scrum*. Addison-Wesley Professional, 2009.

Larman, Craig. *Agile and iterative development: a manager's guide*. Addison-Wesley Professional, 2004.

Kniberg, Henrik. "Kanban vs Scrum." *Crisp AB. Viitattu* 1 (2009): 2011.