IBM Software Group | Rational Software

# Scaling Down Large Projects to Meet the Agile "Sweet Spot"

USC Workshop, March 19-20, 2003

Rational. software

# Presenter

Philippe Kruchten, Ph. D., P. Eng.
Director of Process Development (RUP)
IBM Software Group | Rational Software
638-650 West 41st avenue
Vancouver BC V5Z 2M9
Canada

pbk@rational.com (now)
pbk@ca.ibm.com (future)

# Scaling UP the Agile processes?

- How large can I expand the team and still be agile?
- How long… ?
- How many attributes of agility can I drop and still be agile?
- Which practices can I scale up?
- Which practices I cannot use on large projects?

- Or should we :
  - Understand the ideal conditions of agility
  - Scale down projects in a way that establishes these ideal conditions of agility

***If the mountain will not go to Mahomet, let Mahomet go to the mountain.*** **(Proverb)**

# Agile Process "Sweet Spot"

- High bandwidth communication
  - Small team (10-15), Collocated, Face to face (as opposed to document based)
- Customer on site
  - a customer representative, domain literate and empowered to make decisions
- Short lifecycle (weeks or months, not years)
- Powerful development environment:
  - Rapid development cycle, though automation
  - Continuous integration (builds)
  - Automated test
- Iterative
  - Accommodating change, refactoring
- Business applications (rather than technical, embedded, real-time)
- New development (rather than maintenance)
  - Availability of test regression suite ?

# The “Bitter Spot” (?)

- Large team
- Distributed team
- No empowered customer on site
- Long development cycle
- Inefficient development environment (low level of automation)
- Different cultures

- Low communication bandwidth, reliance on documents
- Waterfall
- Aversion to change

- Try to follow a fixed plan

# Large Project

- Large projects are ***not*** going away
- Tend to :
  - Heavy early planning (and desperately try to follow)
  - Relying solely on written artifacts (workproducts),
    - including in dialog with customer
    - And in following a defined process
- Waterfall approach still dominant paradigm
  - Easy on management
  - “comforting”, false sense of rationality, of determinism
  - Similar to other engineering disciplines

# An Exemplar Large Project

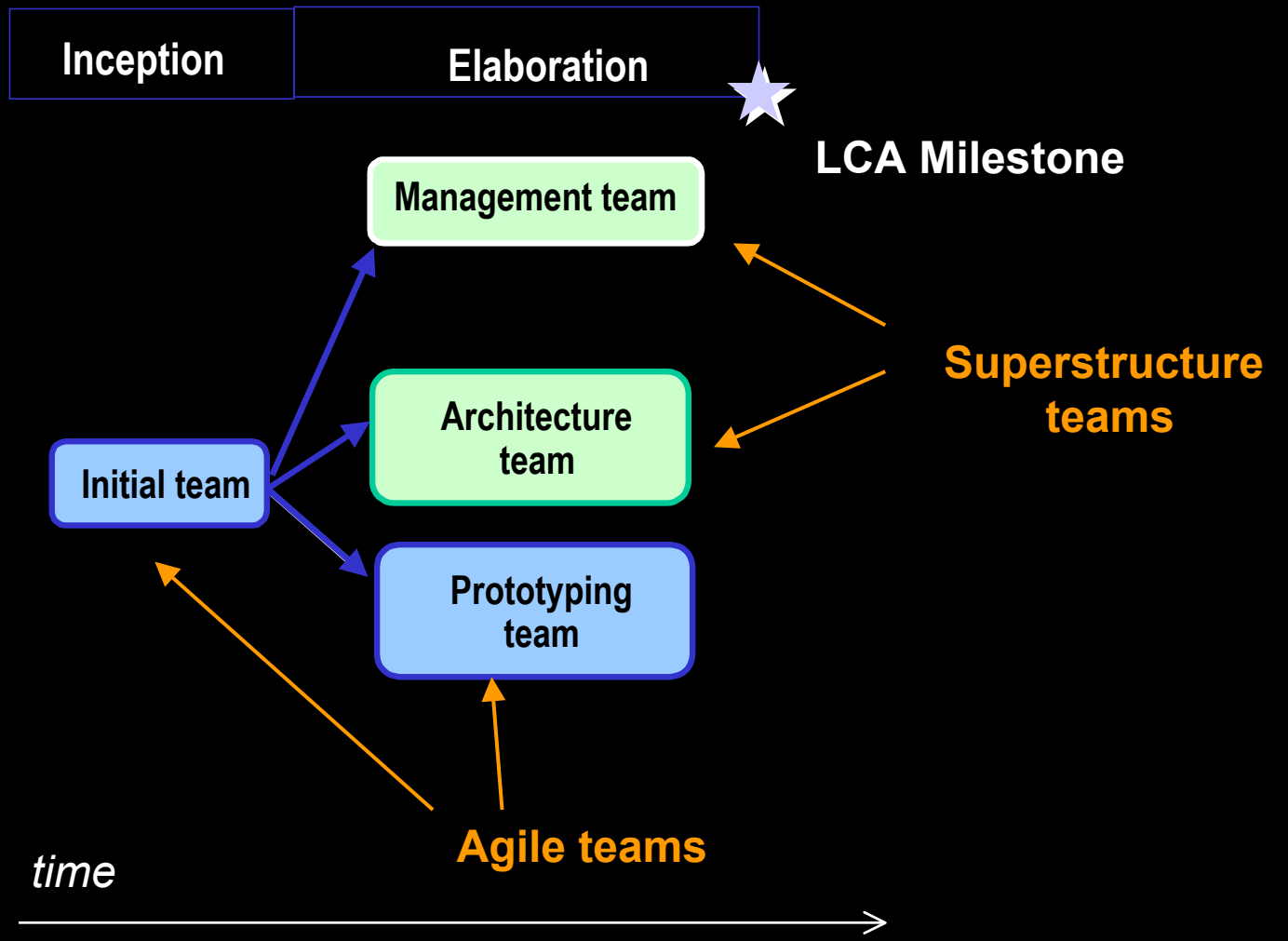
- Avoid the marginal conditions (go from 12 to 18)
- Do not compare a **bad** large project with a **good** small and agile project

- 200 people or more
- Distributed
- Different organizations/companies
- 2 ½ years before first delivery
- Iterative lifecycle
- Good, skilled people
- Access to customer
- Access to efficient programming environment
- New system, architecture not set

# The Ideal Large Project—Agile?

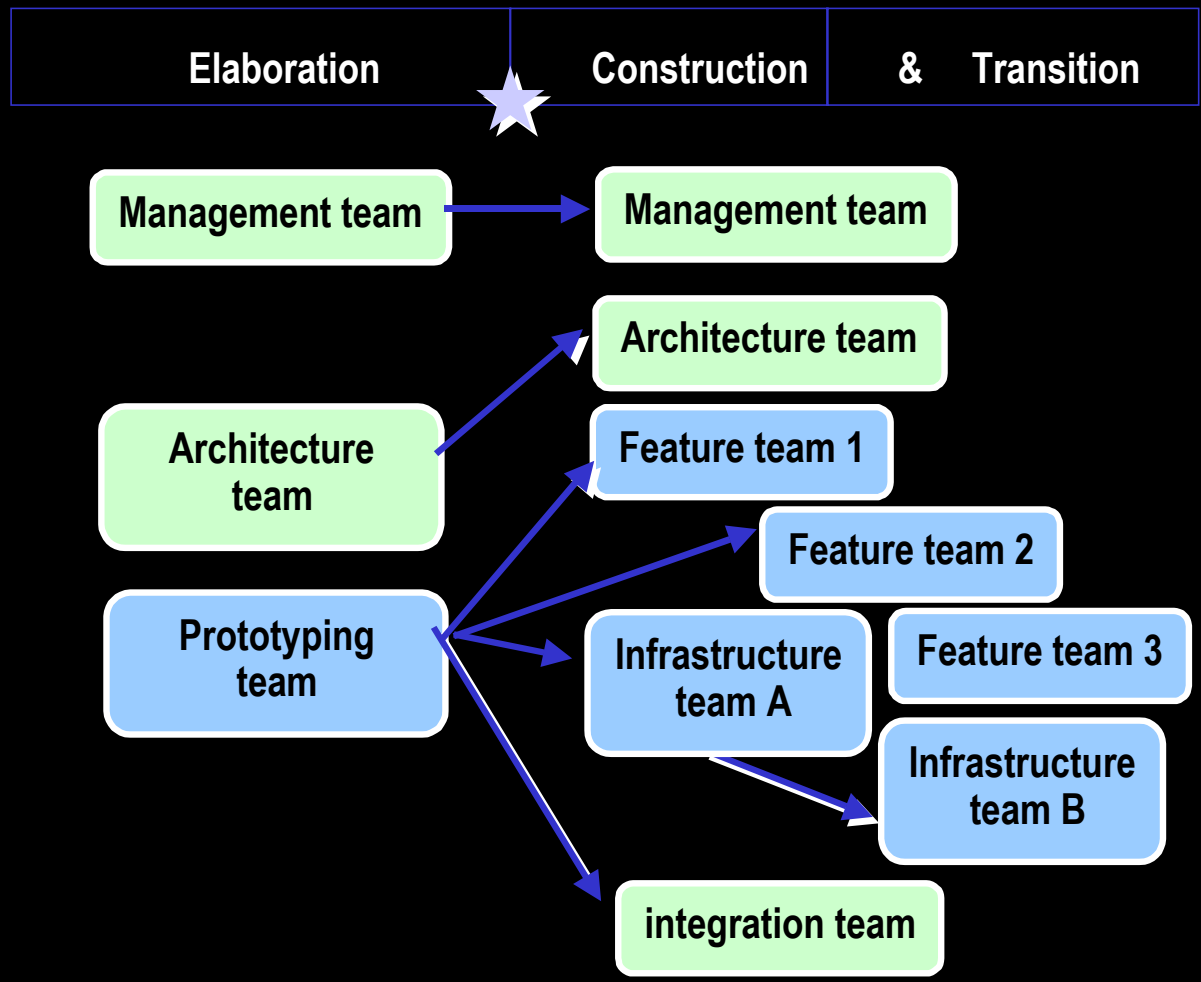
- High bandwidth communication
  - How to achieve beyond 15?
- Customer
  - How to make it available to 150 people
- Focus on results
  - How to focus on something getting out in 2 years?
- Some level of planning and explicit documentation will be necessary
- Break-down the 200 people in:
  - 10 teams of 15
  - Establish the conditions of optimal agility
  - Use the 50 others to fill the space in between, act as the glue
    - Communication
    - Customer role

# Setting up the Large Project—Inception+Elaboration

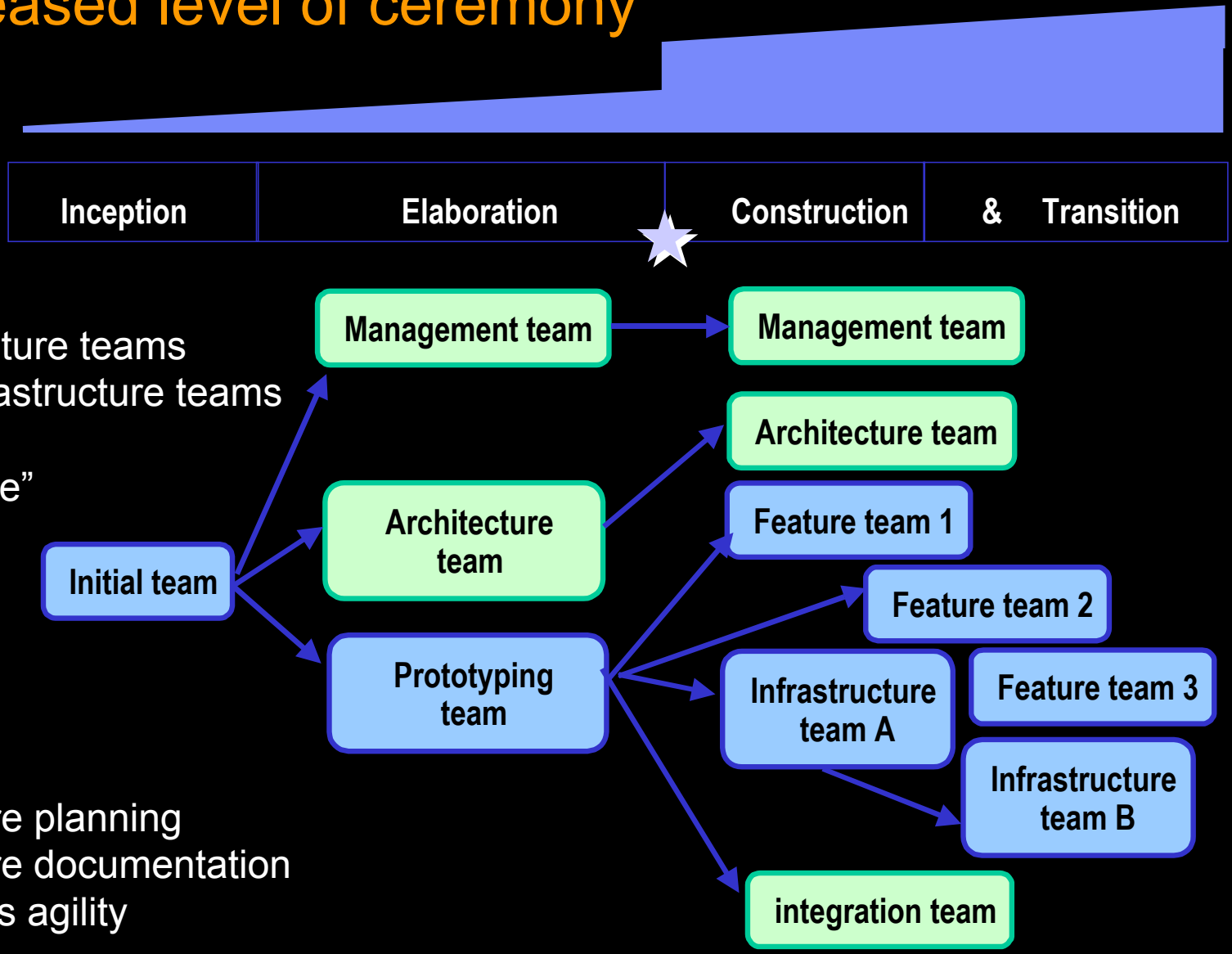


# Setting up the Large Project—Construction+Transition

- Create 2 types of team
  - Feature teams
    - User oriented
  - Infrastructure teams
    - Provider of common services to feature teams



# Increased level of ceremony




- Feature teams
- Infrastructure teams
- "glue"

- More planning
- More documentation
- Less agility

# Where are we?

- Feature teams and infrastructure teams:
  - Organized as “agile projects”
- Added a project superstructure to reproduce ideal conditions
  - Architecture team
  - Integration team
  - Project management team
- Increase in level of ceremony
- More planning involved than with a single agile project
  - Keep feedback loop from iterations, react to change
  - Not a plan first and then execute to plan

# Issues with a Federation of Teams

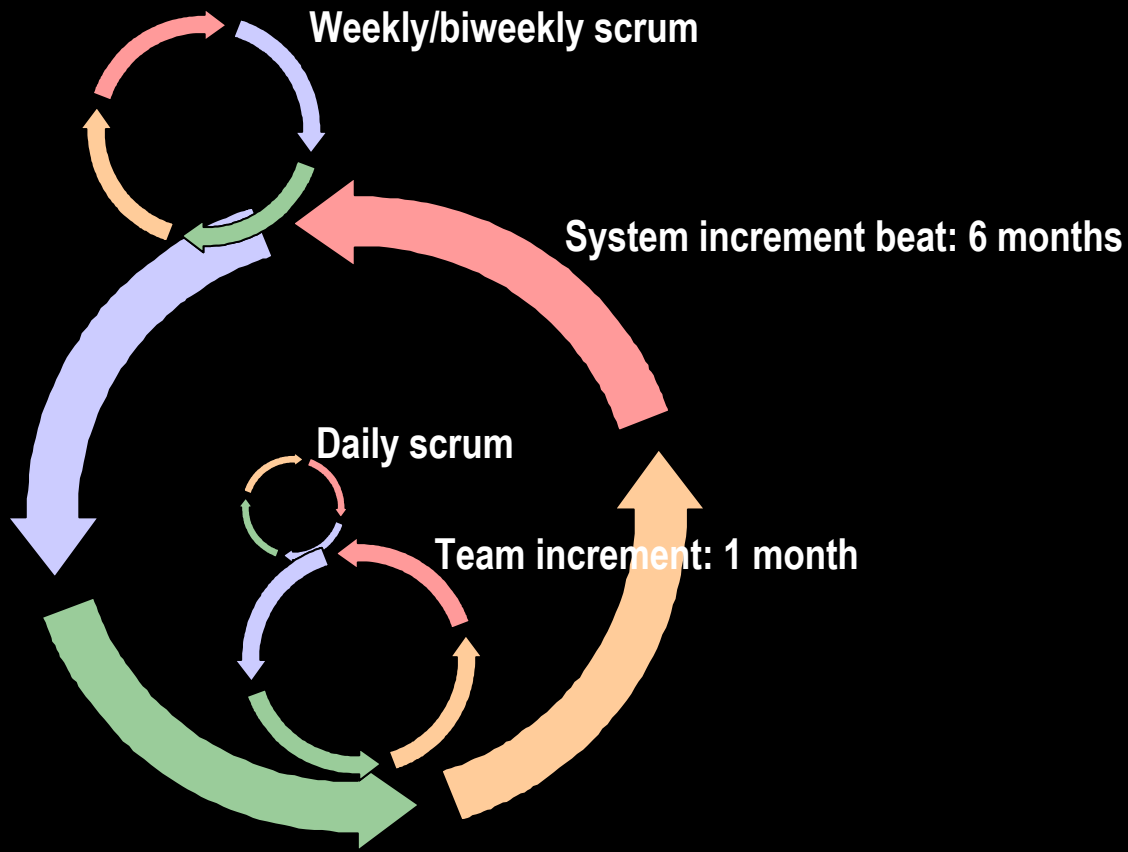
- Dependencies between teams
  - Teams are customer for each other
  - May need “brokering” by architecture team to avoid too much one-to-one communication
- Stovepipes
  - Teams as small fiefdoms, building and empire and reinventing the wheel
  - Architecture team participate in design reviews, planning sessions
- Imbalance: staff and skills
  - Identified by architects, or raised up by team lead
- Communication breakage
- Too much staff too early
- Long cycle: lack of feedback loop?

- Rotation of staff, and circulation of architect
  - “moral” equivalent of pair programming
  - Spreading the culture

# Dual Rhythm

- Long cycle
  - Lack of feed back
  - Team lose track of ultimate goal
  - Need intermediate, tangible goals
- Dual ‘beat’

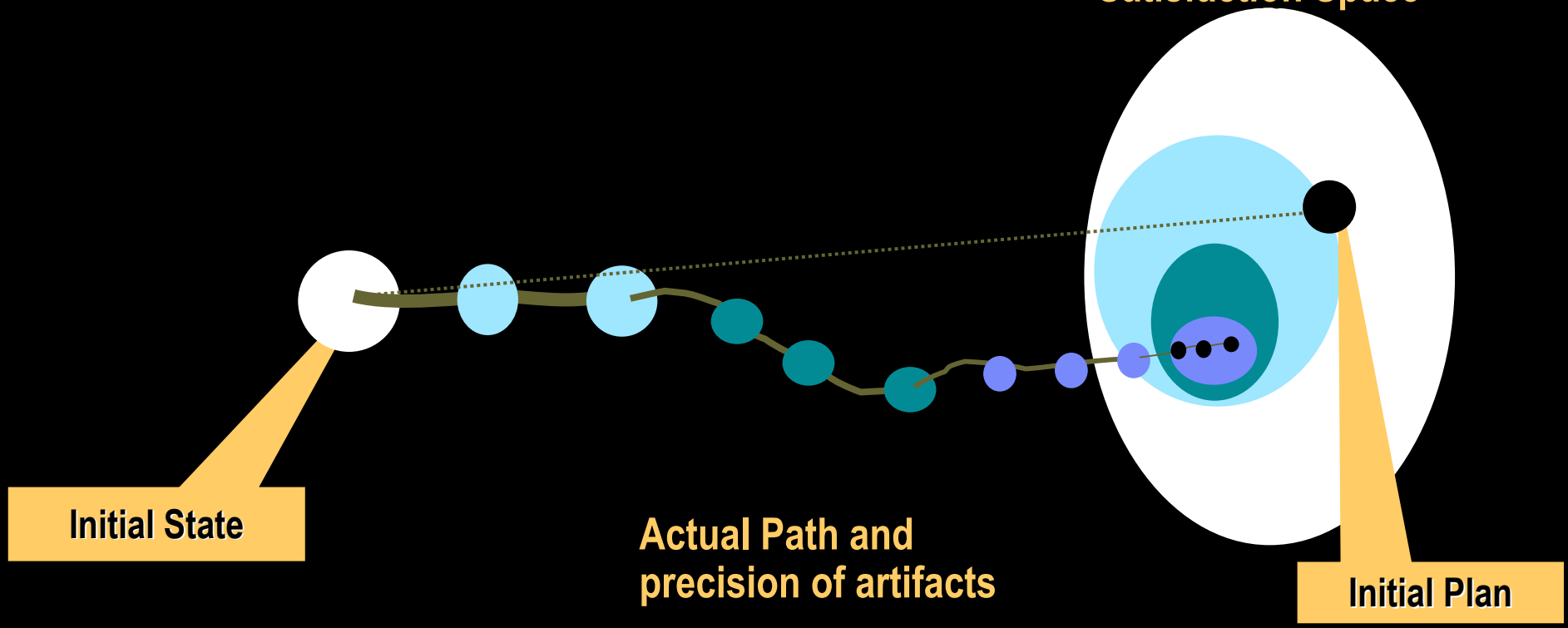
**Example**



# Organizing Project Documentation

- Progressively introducing more explicit documentation
  - More efficient use of staff (e.g., architecture team)
  - Greater visibility
- Software (/System) Architecture
- Project level backlog, Risks and issues
- Requirements: “vision” and key use cases
- Key interfaces
- Recommended practices, project standards (the actual concrete process)
- Reusable elements
- Overall project plan

# Iterations and Demonstrable Progress




**Emergence of Requirements,
Architecture, Design, Plans, and Product**

# How Much Process is Necessary?

**Simple upgrades**
**R&D Prototypes**
**Static web apps**

**Dynamic web apps**
**Packaged applications**
**Component based (J2, .Net)**

**Legacy upgrades**
**Systems of systems**
**Real-time, embedded**
**Certifiable quality**

**Strength of Process**

**When is Less Appropriate?**

- Co-located teams
- Smaller, simpler projects
- Few stakeholders
- ***Early life-cycle phases***
- Internally imposed constraints

**When is More Appropriate?**

- Distributed teams
- Large projects (teams of teams)
- Many stakeholders
- ***Later life-cycle phases***
- Externally imposed constraints
  - Standards
  - Contractual requirements
  - Legal requirements

# Process Strength Over the Life Cycle

**Weak process influence**

**(Optimized for rapid adaptation to change)**

**Strong process influence**

**(Optimized for converging on quality product releases)**

**Product Release**

**Process Weight ➔** $\frac{\text{(Product Quality)}}{\text{(Time to Release)}}$

# Examples

- Ship System 2000 (FS2000)
  - Central software architecture team
  - Iterative development
  - Architecture => blueprint for organization
- Canadian Air Traffic Control system (CAATS)
  - Went further than SS200
  - Customer on-site: large team of actual users of ATC
  - Start prototyping with a small team + arch. team
  - Architecture team “split” at end LCA milestone
    - Played role of facilitator, communication vertex
  - Integration team
  - Progressive introduction of explicit documentation

# Examples (cont.)

- Rational Software's Product Group
  - Team of teams (700 total)
  - Distributed – 7 sites around the globe
  - Varied culture (acquisition)
  - Centralized:
    - Architecture
    - Product definition, with "local rep"
    - Integration (installer, licensing, etc.)
  - Dual rhythm
    - Major beat: 6 months
    - Internal beat: monthly

# Summary

- Large projects set up as as a **federation of agile teams**
- **Two levels** of:
  - Communication
  - Organization
  - Project ‘beat’
- **Software architecture** serves as the blueprint for the team structure
  - “Emerges” during elaboration with a small team (or two)
- Then **Architecture team + Integration team** add **communication “glue”**
  - Serve as customers
  - Facilitate communication
  - Balance skills, load
  - Foster reuse
  - Assemble final product
- **Gradual introduction of explicit documents**
  - Where they support effective communication, reduce ambiguity, spread common culture

# References and further reading


1. P. B. Kruchten, *The Rational Unified Process: An Introduction, 2 ed*. Boston: Addison-Wesley, 2000.
2. B. W. Boehm, D. Port, M. Abi-Antoun, and A. Egyed, "Guidelines for the Life Cycle Objectives (LCO) and the Life Cycle Architecture (LCA) deliverables for Model-Based Architecting and Software Engineering (MBASE)," USC, Los Angeles, USC Technical Report USC-CSE-98-519, 1999.
3. K. Schwaber and M. Beedle, *Agile Software Development with SCRUM*. Upper Saddle River, NJ: Prentice-Hall, 2002.
4. L. Brownsword and P. Clements, "A Case Study in Successful Product Line Development," Software Engineering Institute, CMU/SEI-96-TR-035, 1996.
5. T. Paine, P. Kruchten, and K. Toth, "Modernizing Air Traffic Control through Modern Software Methods," in *38th Annual Air Traffic Control Association Conference*. Nashville, Tenn.: ATCA, 1993.
6. P. Kruchten, "Iterative Software Development for Large Ada Programs," in *Proc. Ada-Europe conference, Montreux, Switzerland*, vol. LNCS 1088, A. Strohmeier, ed.: Springer-Verlag, 1996, pp. 101-110.
7. P. Kruchten and C. J. Thompson, "An Object-Oriented, Distributed Architecture for Large Scale Systems," in *Proc. of Tri-Ada'94*, Baltimore, November 1994: ACM.
8. P. Kruchten, "The Software Architect, and the Software Architecture Team," in *Software Architecture*, P. Donohue, ed. Boston: Kluwer Academic Publishers, 1999, pp. 565-583.