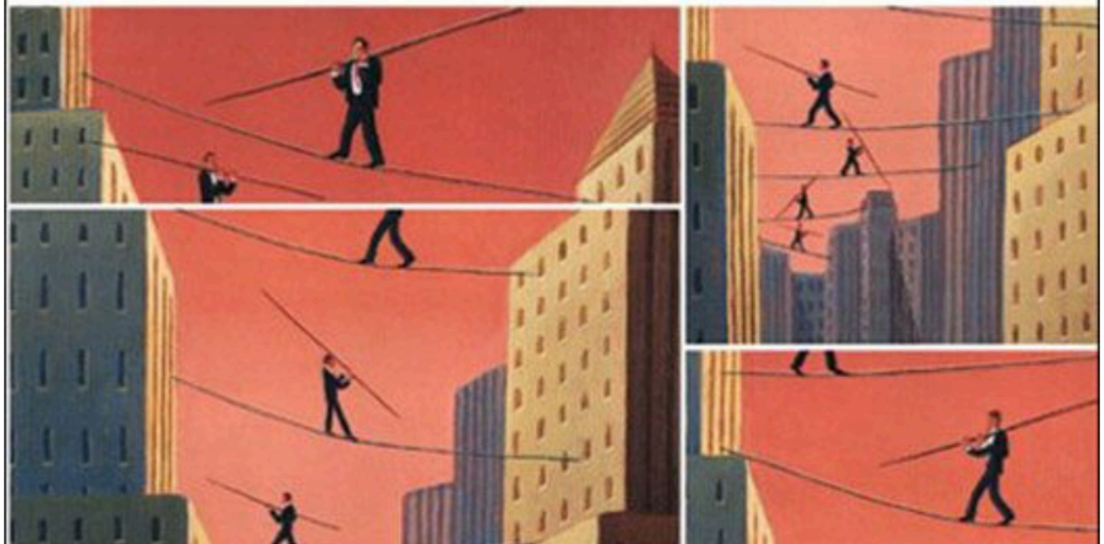
Architecture and Lifecycles

Barry Boehm
Richard Turner



Balancing Agility and Discipline

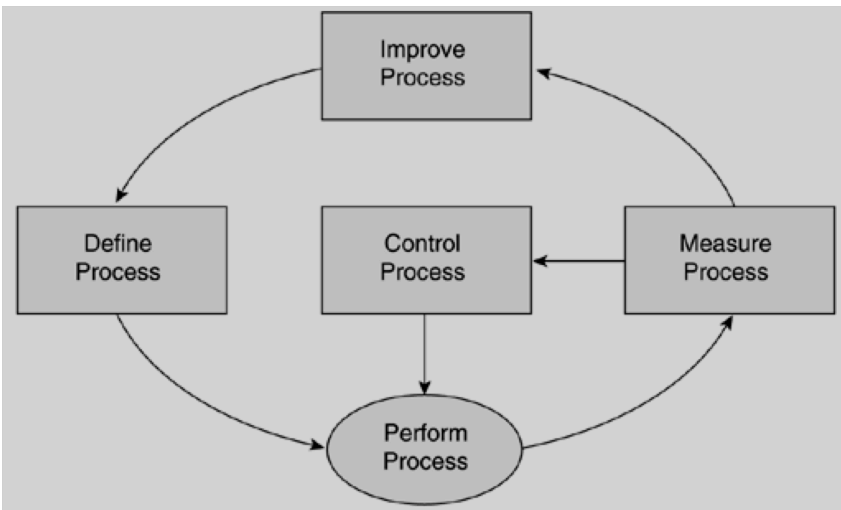
A Guide for the Perplexed

Forewords by
Grady Booch · Alistair Cockburn · Arthur Pyster

Balancing Agility and Discipline

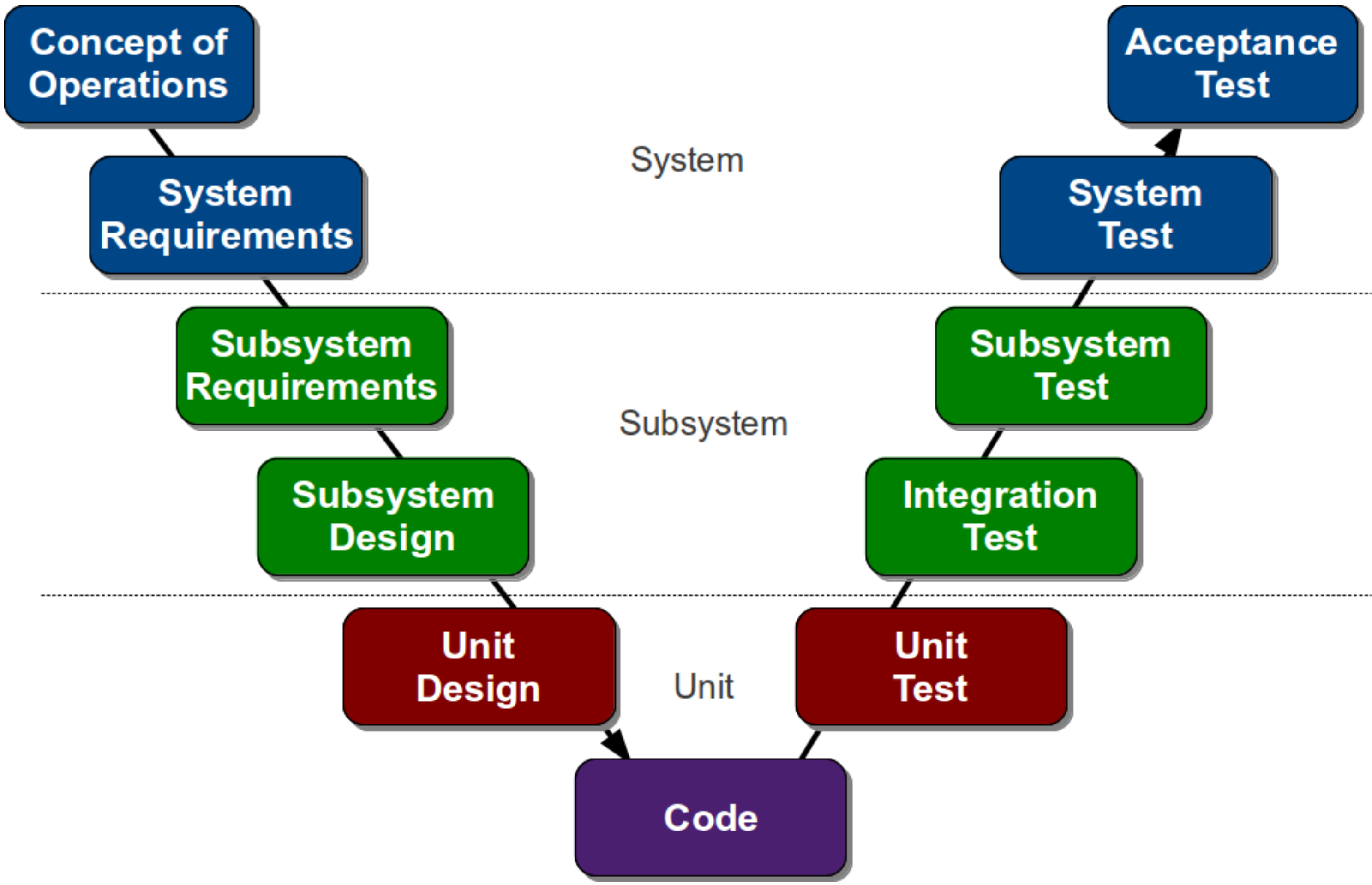
- Lifecycles generally impose some discipline on the development process.
- Software Architectures often feature in Lifecycles as a stage or support for analysis or design
- Lifecycles exist because they codify useful patterns of activity and save us time and effort
- Agility focusses on getting adequate solutions to stakeholders with less time and effort
- We need to balance the discipline of lifecycles against the delivery focus of agility
- Boehm and Turner explore this balance in their book

Lifecycles



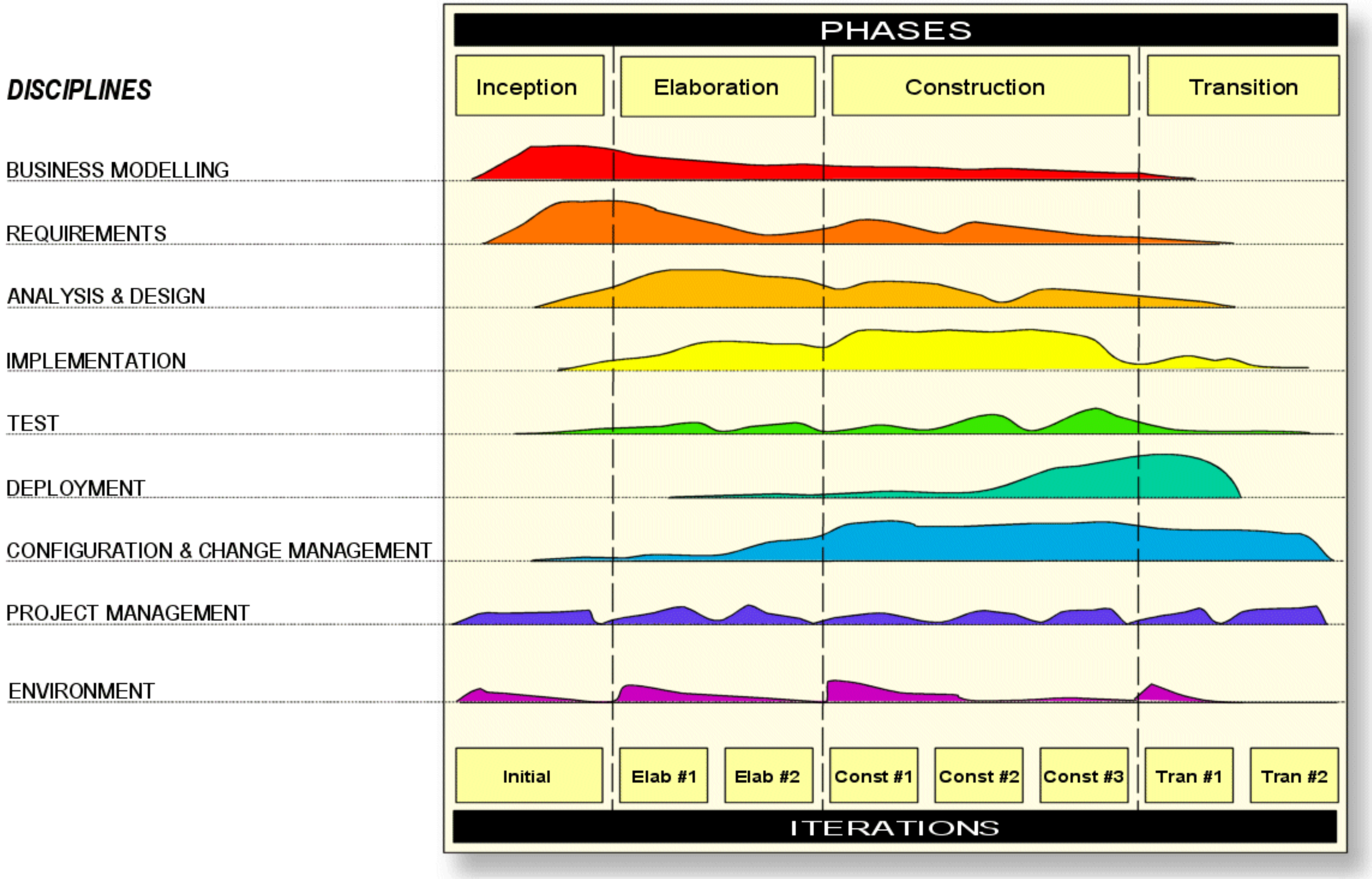
- Lifecycles underpin development processes by ordering stages and activities.
- Any good organisation is always looking to improve its processes so there is usually an ongoing process improvement cycle focussed on making the process better.

Classical: V-Model



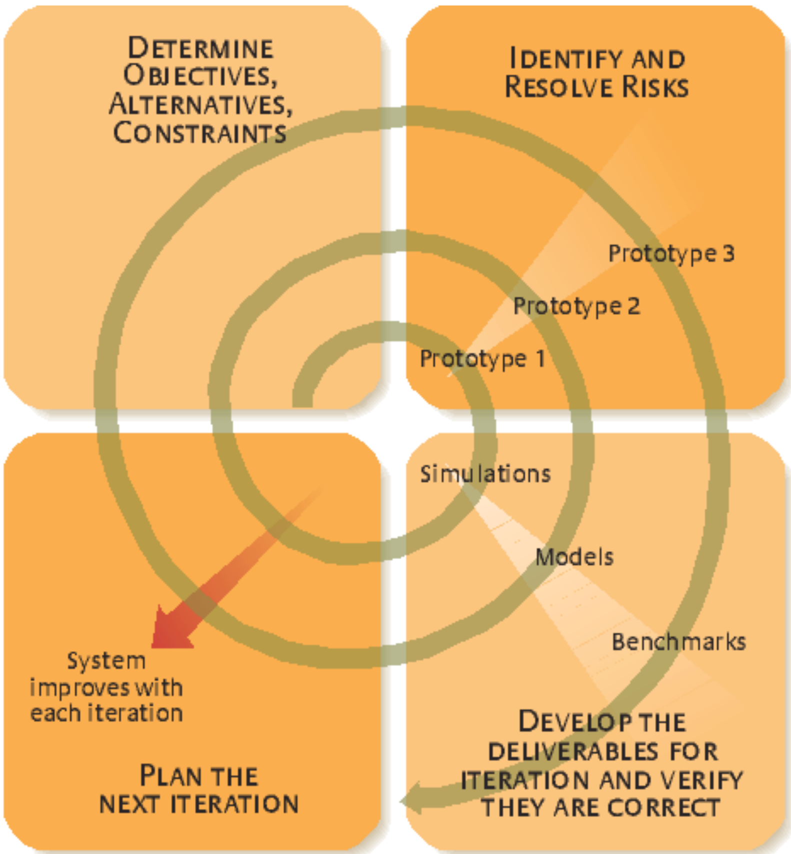
https://davenicolette.files.wordpress.com/2012/02

Classical: RUP



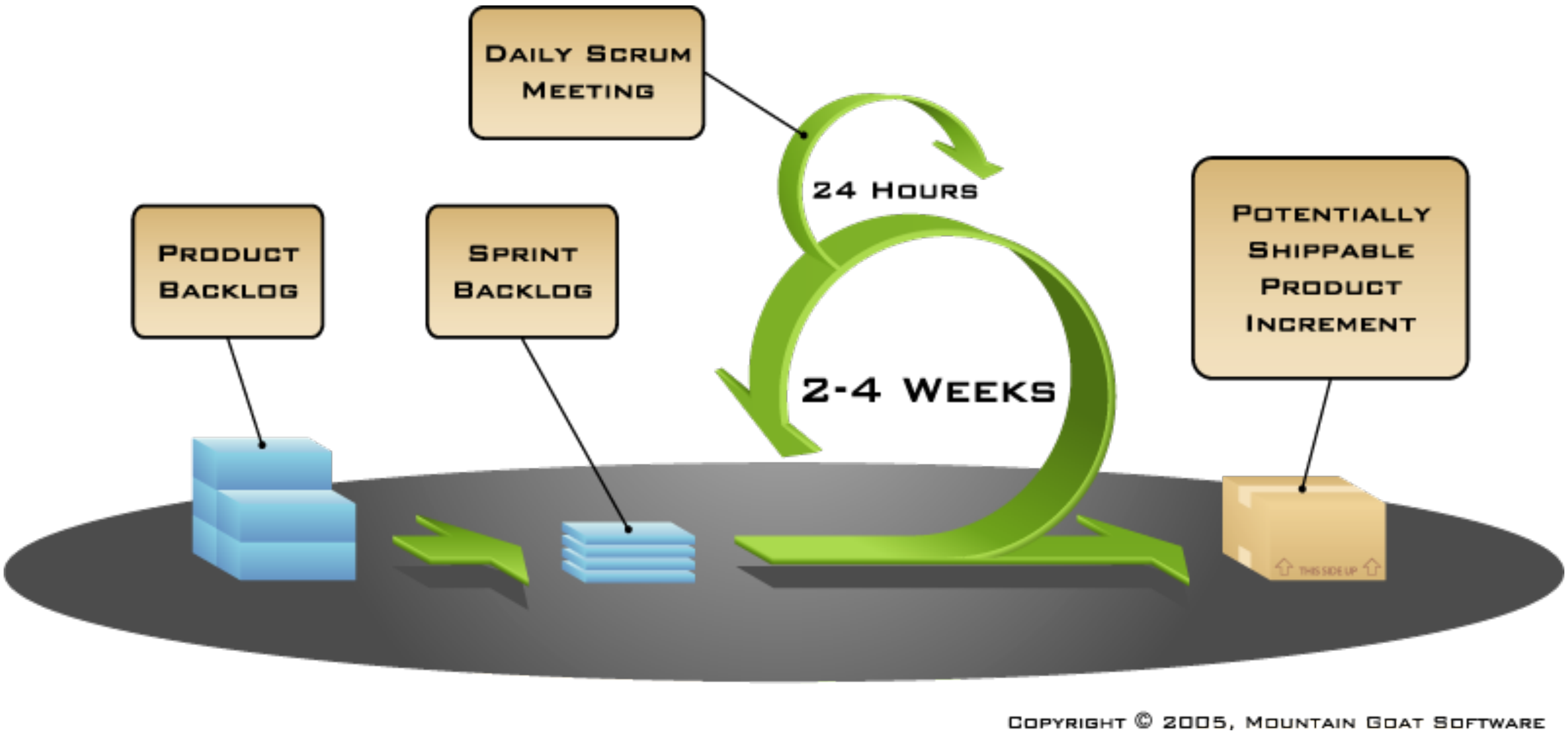
https://davenicolette.files.wordpress.com/2012/02/rup.png

Transitional: Spiral Model



Adapted from B. Boehm, "A Spiral Model of Software Development and Enhancement," ACM SIGSOFT Engineering Notes 11, no. 4 (1986): 25.

Agile: SCRUM



Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment
Copyright © 2005, Mountain Goat Software

Agile Manifesto

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

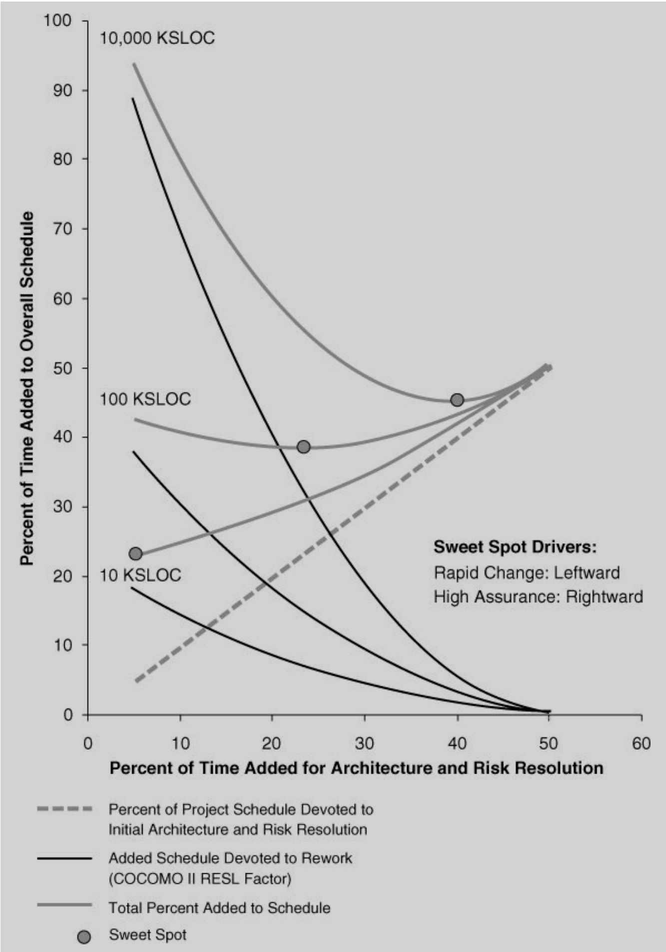
Individuals and interactions	over	processes and tools
Working software	over	comprehensive documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan

That is, while there is value in the items on the right, we value the items on the left more. [agilemanifesto.org]

Agile Practice

- Test-first programming
- Refactoring
- Continuous integration
- Simple Design
- Pair Programming
- Common Codebase
- Coding Standards
- Open Work Area

Bigger needs more structure



From Boehm and Turner

Architecture, Agility and Structure

Characteristics	Agile	Plan Driven
Application		
Primary goals	Rapid value; responding to change	Predictability, stability, high assurance
Size	Smaller teams and projects	Larger Teams and projects
Environment	Turbulent; high-change; project-focus	Stable; low-change; project/ organisation focus

Architecture, Agility and Structure

Characteristics	Agile	Plan Driven
Management		
Customer relations	Dedicated on-site customers; focus on prioritized increments	As-needed customer interactions; focus on contract provisions
Planning and Control	Internalized plans; qualitative control	Documented plans; quantitative control
Communication	Tacit interpersonal knowledge	Explicit documented knowledge

Architecture, Agility and Structure

Characteristics	Agile	Plan Driven
Technical		
Requirements	Prioritized informal stories and test cases; undergoing unforseeable change	Formalized project, capability, interface, quality, forseeable evolution requirements
Development	Simple design; short increments; refactoring assumed to be inexpensive	Extensive Design; longer increments; refactoring assumed expensive
Testing	Executable test cases define requirements	Documented test plans and procedures

Architecture, Agility and Structure

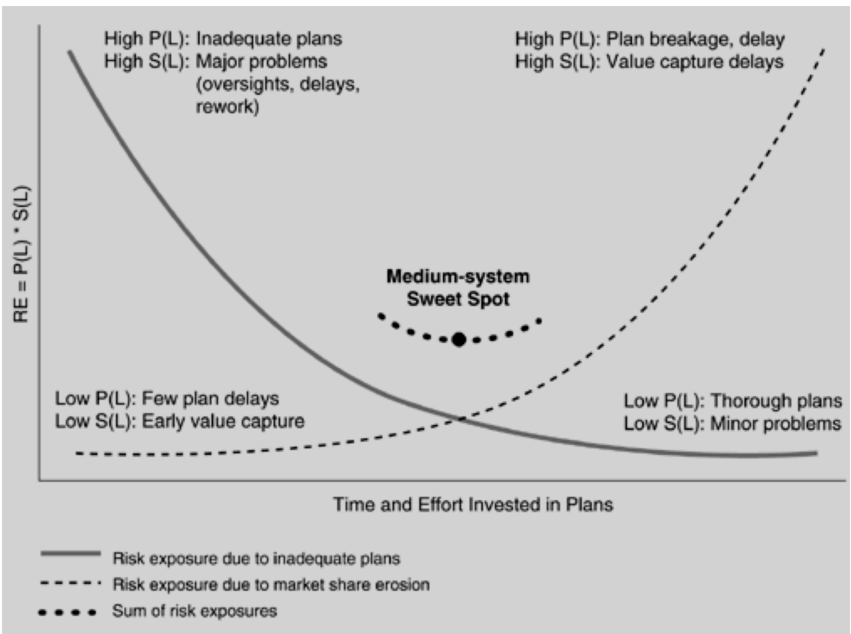
Characteristics	Agile	Plan Driven
Personnel		
Customers	Dedicated, collocated CRACK* performers	CRACK* performers – not always collocated
Developers	High level skills required throughout – less scope for lower performers	High skill people needed at critical points, more possibility to use lower skilled people
Culture	Comfort and empowerment through freedom (chaos?)	Comfort and empowerment through policies and procedures (order).

Architecture, Agility and Structure

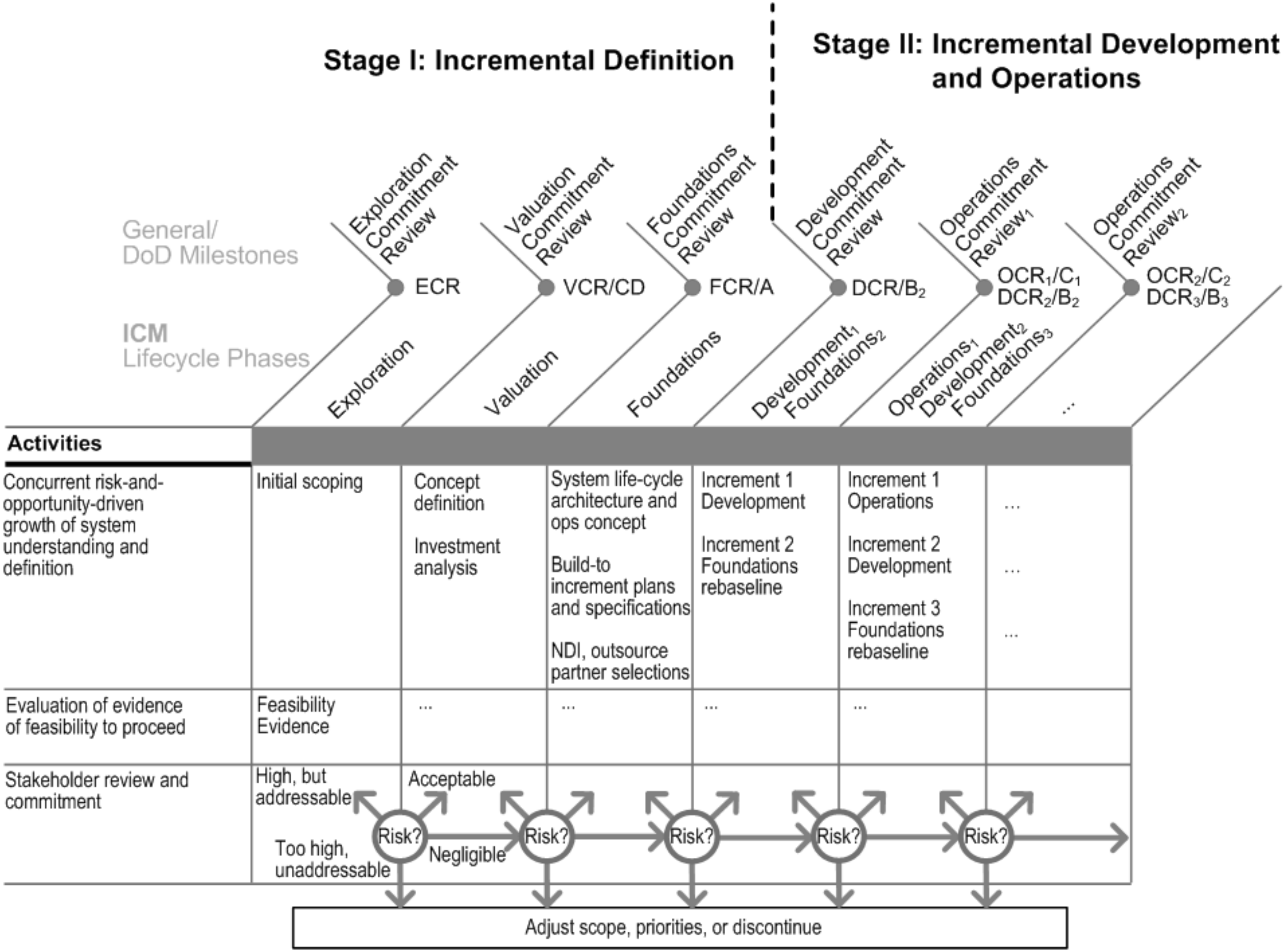
- Work top-down and bottom-up simultaneously – balance will depend on the size and complexity of the project.
- Top-down does architectural workbased on things like patterns.
- Bottom-up develops implementation and environment-specific constraints and solutions.
- Focus on QAs, scenarios, tactics and processes to reconcile competing aspects provides a bottom-up/top-down link
- Balancing commitment and flexibility

Factor	Agility Discriminators	Plan-Driven Discriminators
Size	Well-matched to small products and teams. Reliance on tacit knowledge limits scalability.	Methods evolved to handle large products and teams. Hard to tailor down to small projects.
Criticality	Untested on safety-critical products. Potential difficulties with simple design and lack of documentation.	Methods evolved to handle highly critical products. Hard to tailor down to low-criticality products.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments, but a source of potentially expensive rework for highly stable environments.	Detailed plans and Big Design Up Front excellent for highly stable environment, but a source of expensive rework for highly dynamic environments.
Personnel	Requires continuous presence of a critical mass of scarce Cockburn Level 2 or 3 experts. Risky to use non-agile Level 1B people.	Needs a critical mass of scarce Cockburn Level 2 and 3 experts during project definition, but can work with fewer later in the project —unless the environment is highly dynamic. Can usually accommodate some Level 1B people.
Culture	Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom. (Thriving on chaos)	Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear policies and procedures. (Thriving on order)

Risk links Plans and Agility



- P(L) – the probability of loss.
- S(L) – the size of loss
- These curves vary depending on the type of project.



Incremental Commitment Model: http://csse.usc.edu/csse/research/ICM/

Incremental Commitment Model

- Boehm, Barry, and Jo Ann Lane. "Using the incremental commitment model to integrate system acquisition, systems engineering, and software engineering." *CrossTalk* 19.10 (2007): 4-9.
- Koolmanojwong, Supannika, and Barry Boehm. "The incremental commitment model process patterns for rapid-fielding projects." *New Modeling Concepts for Today's Software Processes*. Springer Berlin Heidelberg, 2010. 150-162.
- Boehm, Barry. "Applying the Incremental Commitment Model to Brownfield Systems Development." *Proceedings, CSER* (2009).

Architecture, Analysis and Lifecycle

- Kazman, R., R. L. Nord, and M. Klein. "A Life-Cycle View of Architecture Analysis and Design Methods (CMU/SEI-2003-TN-026). Pittsburgh, PA: Software Engineering Institute." (2003).
- Kazman *et al* look at how analysis techniques fit into lifecycle stages.
- They refer to several analysis techniques and then outline how they contribute to lifecycle stages.

Analysis Techniques

Acronym	Name	Inputs	Outputs
QAW	Quality Attribute Workshop	Mission drivers; system architectural plan	Raw scenarios; consolidated scenarios; priorities; refined scenarios
ADD	Attribute Driven Design	Constraints; functional and QA requirements	Module; component; and deployment architectural views
ATAM	Architecture Tradeoff Analysis Method	Business/mission drivers; existing architecture documentation	Potential arch approaches; scenarios; QA questions; risks; themes; sensitivities; tradeoffs
CBAM	Cost-benefit analysis method	As ATAM + existing scenarios	Architectural strategies; priorities; risks
ARID	Active Reviews for Intermediate Designs	Seed scenarios; existing Arch documentation	Issues and problems for Architecture

Analysis Techniques and Stage

Life-Cycle Stage	QAW	ADD	ATAM	CBAM	ARID
Business needs and constraints	Input	Input	Input	Input	
Requirements	Input; output	Input	Input; output	Input; output	
Architecture design		Output	Input; output	Input; output	Input
Detailed design					Input; output
Implementation					
Testing					
Deployment					
Maintenance				Input; output	

Lifecycle Stages and Architecture Activity

Life-Cycle Stage	Architecture-Based Activity
Business needs and constraints	• Create a documented set of business goals: issues/environment, opportunities, rationale, and constraints using a business presentation template.
Requirements	• Elicit and document six-part quality attribute scenarios using general scenarios, utility trees, and scenario brainstorming.
Architecture design	• Design the architecture using ADD. • Document the architecture using multiple views. • Analyze the architecture using some combination of the ATAM, ARID, or CBAM.
Detailed design	• Validate the usability of high-risk parts of the detailed design using an ARID review.
Implementation	
Testing	
Deployment	
Maintenance	• Update the documented set of business goals using a business presentation template. • Collect use case, growth, and exploratory scenarios using general scenarios, utility trees, and scenario brainstorming. • Design the new architectural strategies using ADD. • Augment the collected scenarios with a range of response and associated utility values (creating a utility-response curve); determine the costs, expected benefits, and ROI of all architectural strategies using the CBAM. • Make decisions among architectural strategies based on ROI, using the CBAM results.

Summary

- There are many possible lifecycles and variants on these lifecycles.
- For any particular area of activity we need to find the balance between agility and discipline.
- Risk links discipline and agility
- QAs, scenarios and tactics help link architecture to more agile practice.
- Architectural analysis techniques provide useful information for lifecycle activities however they are arranged in a process.
- Processes like the Spiral Model or ICM provide processes that are sensitive to project risk.