Course "Softwareprozesse"

Agile Manifesto

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik

- Origin of agile
- Agile Manifesto
  - core part, principles list, background
  - common misunderstanding
  - culture
- Underlying assumptions
- Classical view on agile
  - Too much vs. too little planning
  - XP planning game
- Usage frequency of various agile methods

Learning objectives

- Understand the key ideas of the agile movement
- Be able to detect those ideas at work in processes
- Be able to detect deviations and violations of these ideas

On the origins of Agile: The Waterfall myth

It is often said that

Winston W. Royce: "Managing the development of large software systems", Proc. WESCON, 1970

has proposed the proper way to develop software as this:

(a plan-driven approach)

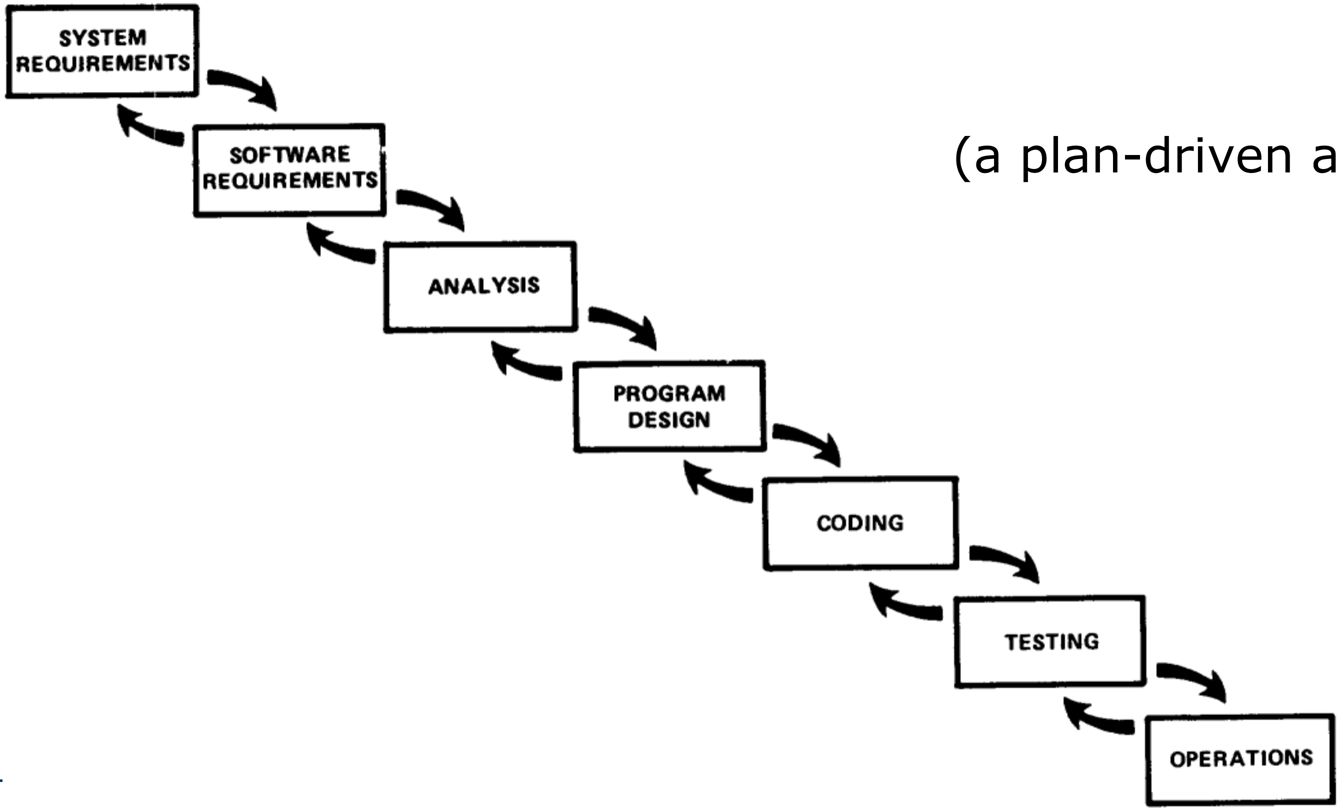


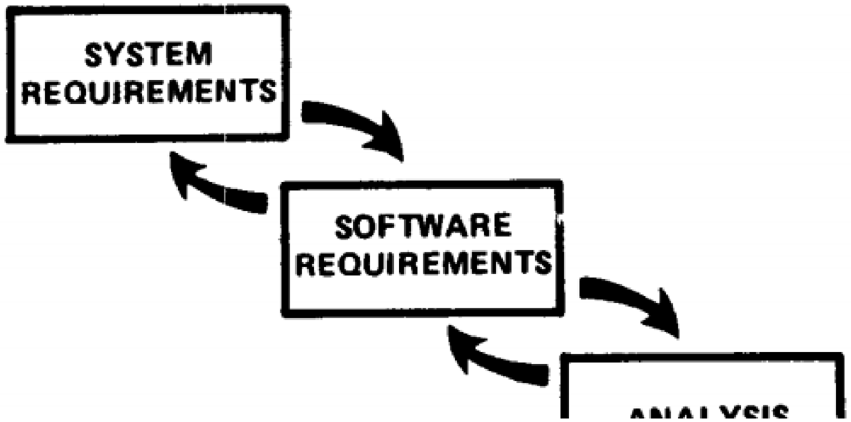
Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

On the origins of Agile: The Waterfall myth (2)

- What it really said:
  - *"In my experience, [this method] has never worked on large software development efforts"*
  - *"To give the contractor free rein between requirement definition and operation is inviting trouble."*
- Royce recommends several **additions as a repair**:
    - **Note**: The article talks about *"spacecraft mission planning, commanding and post-flight analysis"* where requirements tend to be well-understood and stable!
  - *"preliminary design"*: Architectural design and its documentation
  - *"plan, control and monitor testing"*: approach QA systematically
    - and take it very seriously
  - *"do it twice"*: iterate; build a throw-away prototype
    - mostly to understand non-functional behavior
  - *"involve the customer"*: get feedback
    - have the customer commit to parts of the effort before final delivery

On the origins of Agile: Waterfall practice

- Subsequent practitioners have often used the basic plan-driven waterfall process with only two of the repairs:
  - Architectural design; Systematic testing
- but missed two crucial ones:
  - **iterative development**; **feedback/incremental commitment**
- Furthermore, they have often overlooked a crucial element for information systems:
  - **User requirements** are needed before system requirements
    - User requirements are situated in the problem domain (➔ value!)
    - System requirements are situated in the solution domain (HW/SW)



On the origins of Agile: Effect of the misunderstanding

- As a result, many SW projects got into huge trouble
  - massive plan overruns; sometimes no useful SW produced at all
  - especially for projects with unclear requirements
- As computers got faster, some practitioners developed better ways of developing SW
  - highly iterative; fewer documents; automated testing
- Other projects at least used iterations to reduce requirements risks
  - but many projects continued to work badly
    - Conjecture: too much in the classical-view culture made it hard to see what process was needed.
- Then in 2001, 17 practitioners got together and formulated their worldview in the Agile Manifesto
  - since then, the modern view has quickly gained acceptance
    - at least seemingly

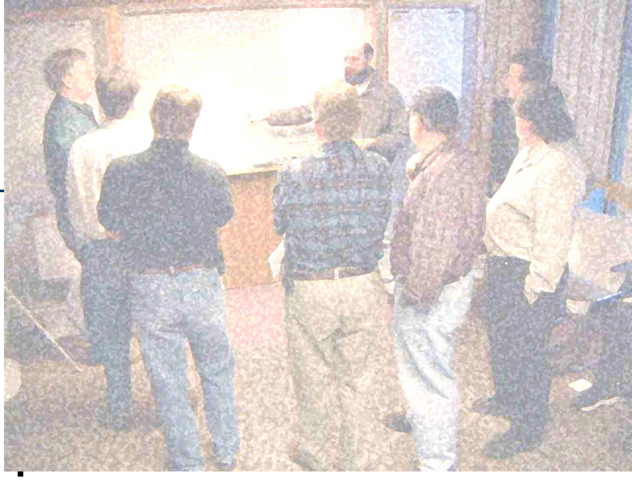
Prelude: Agile is about priorities

- Jim Highsmith, Alistair Cockburn: "*Agile Software Development: The Business of Innovation*", IEEE Computer, September 2001
  - Very good introduction into the agile way of thinking
- *"Processes, tools, documentation, contracts, and plans are useful.*
  But when push comes to shove — and it usually does — something must give, and we need to be clear about what stays and what gives."
  - (Agile methods were initially called "lightweight" methods, because they attempted to get rid of dispensable process elements)

Highsmith

Cockburn

The Agile Manifesto



http://www.agilemanifesto.org (2001)

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

  (this stays) (this gives way)

  That is, while there is value in the items on the right, we value the items on the left more."

Plausible?
When?

Agile Manifesto: Principles

[We discuss: When is this a good idea? When is it not? Elements of modern view?]

We follow these principles:

1. Our highest priority is to satisfy the customer
   - through early and continuous delivery of valuable software.

   (Value focus, not quality focus!)
2. Welcome changing requirements, even late in development.
   - Agile processes harness change for the customer's competitive advantage.

   "agile"
3. Deliver working software frequently,
   - from a couple of weeks to a couple of months, with a preference to the shorter timescale.

   (center pillar)
4. Business people and developers must work together daily
   - throughout the project.

Agile Manifesto: Principles (2)

[We discuss: When is this a good idea? When is it not? Elements of modern view?]

5. Build projects around motivated individuals.
   - Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.
   - The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

"lightweight"

Agile Manifesto: Principles (2)

[We discuss: When is this a good idea? When is it not? Elements of modern view?]

9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity, the art of maximizing the work not done, is essential.

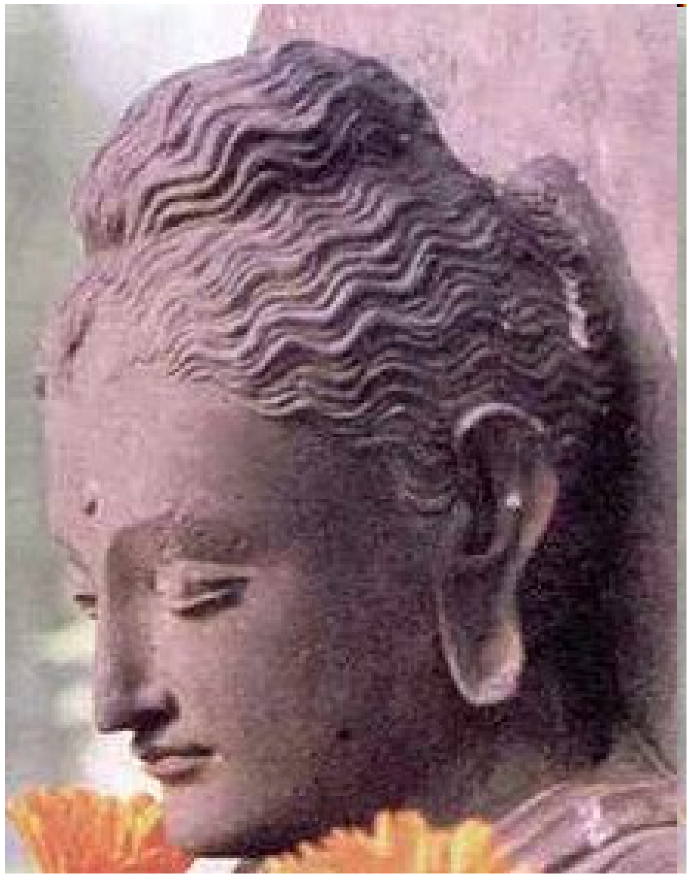
(interesting one!)

11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective,
    - then tunes and adjusts its behavior accordingly.

Use your common sense!

Believe nothing,
no matter where you read it
or who said it,
not even if I have said it,
unless it agrees with your own reason
and your own common sense.

Gautama Siddharta Buddha



The still-common misunderstanding

Source: http://agilemanifesto.org/history.html

- "The Agile movement is not anti-methodology,
  - in fact, many of us want to restore credibility to the word methodology.
- We want to restore a balance.
  - **We embrace modeling**, but not in order to file some diagram in a dusty corporate repository.
  - **We embrace documentation**, but not hundreds of pages of never-maintained and rarely-used tomes.
  - **We plan**, but recognize the limits of planning in a turbulent environment."

- Agile Methods are about clear positions in difficult trade-offs
  - not about letting all discipline and process go and just hack.

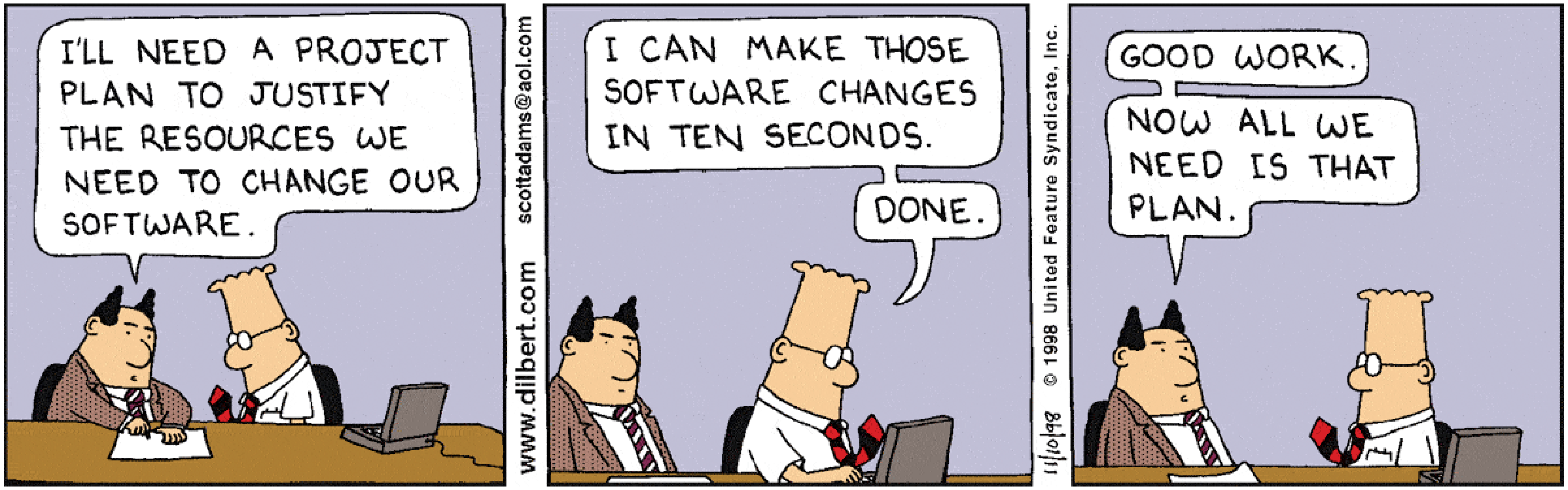
The manifesto folks on culture

Source: http://agilemanifesto.org/history.html

- "we all felt privileged to work with a group of people who held a set of compatible values,
  - a set of values based on trust and respect for each other and promoting organizational models based on people, collaboration, and building the types of organizational communities in which we would want to work."

- "In order to succeed in the new economy, to move aggressively into the era of […] the web,
  - companies have to rid themselves of their Dilbert manifestations of make-work and arcane policies."
    - Does anybody here not know Dilbert?

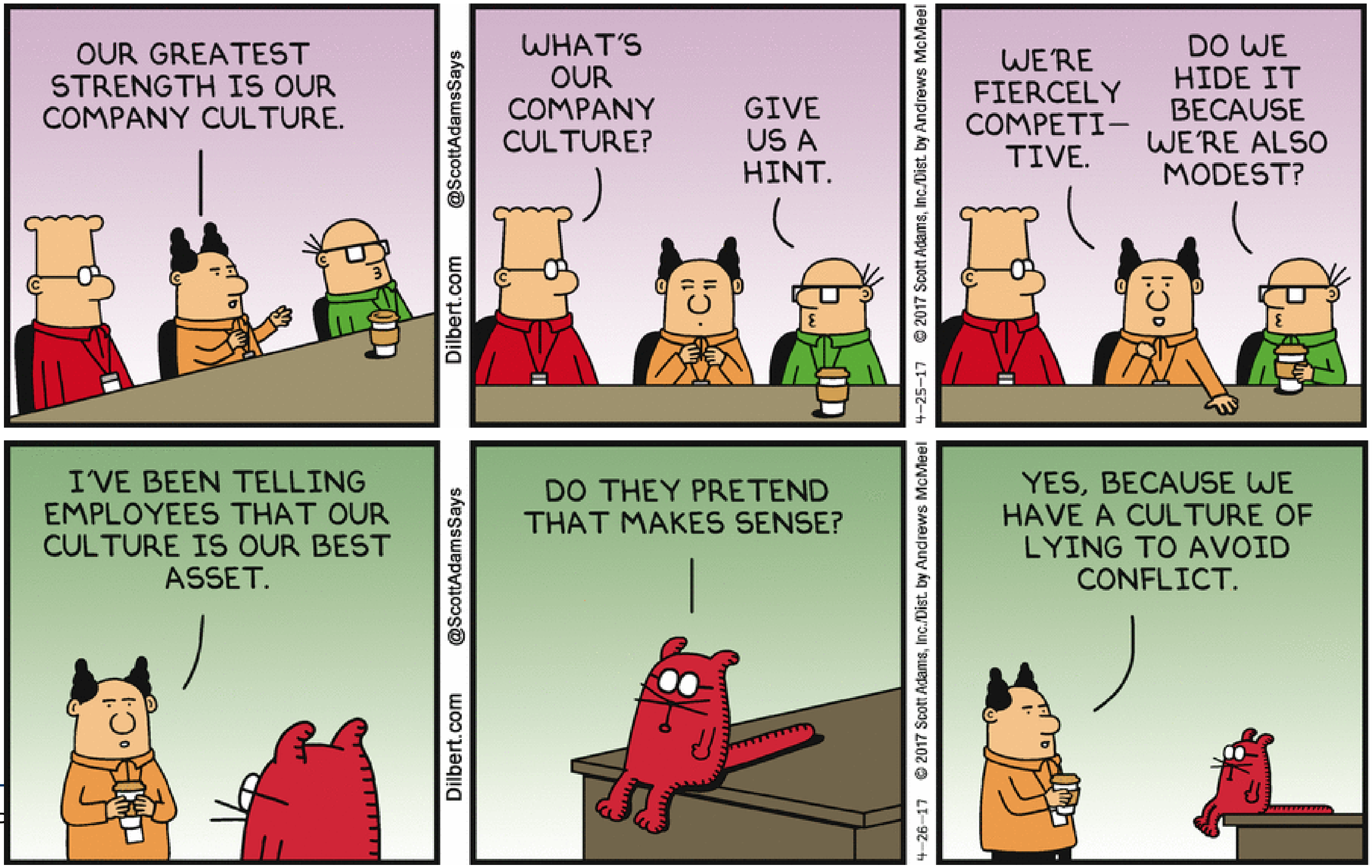
Dilbert example


I'LL NEED A PROJECT PLAN TO JUSTIFY THE RESOURCES WE NEED TO CHANGE OUR SOFTWARE.
www.dilbert.com scottadams@aol.com
I CAN MAKE THOSE SOFTWARE CHANGES IN TEN SECONDS.
DONE.
11/10/98 © 1998 United Feature Syndicate, Inc.
GOOD WORK.
NOW ALL WE NEED IS THAT PLAN.

https://dilbert.com/strip/1998-11-10

The manifesto folks on culture (2)

- "[…] in the final analysis, the meteoric rise of interest in—and sometimes tremendous criticism of—Agile Methodologies is about the mushy stuff of values and culture."


OUR GREATEST STRENGTH IS OUR COMPANY CULTURE.
WHAT'S OUR COMPANY CULTURE?
GIVE US A HINT.
WE'RE FIERCELY COMPETI-TIVE.
DO WE HIDE IT BECAUSE WE'RE ALSO MODEST?
I'VE BEEN TELLING EMPLOYEES THAT OUR CULTURE IS OUR BEST ASSET.
DO THEY PRETEND THAT MAKES SENSE?
YES, BECAUSE WE HAVE A CULTURE OF LYING TO AVOID CONFLICT.
Dilbert.com @ScottAdamsSays
4-25-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel
4-26-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel

https://dilbert.com/strip/2017-04-25
https://dilbert.com/strip/2017-04-26

Assumptions of Agile Methods

1. You cannot foresee the development of a whole project
2. Developers are motivated, technically competent, and capable of good judgement
    - if they do not have to work in Dilbert-like work environments
3. A project can adjust to unforeseen changes
    - In particular, changes to the design are not ruinously expensive
4. Adjustment is easier if everything that need not be in writing, is not done in writing
    - Code must be in writing
    - Some documentation may, too (e.g. for certification)
    - Most other information does not.
        - And where it does, short-lived throw-away writing is often sufficient.

- Agile methods are useful if these assumptions are met,
    - they are problematic if even one of them is not.
    - The most risky assumption is that of cheap design changes

When are the assumptions true?

1. "You cannot foresee the development of a whole project"
   - Almost always true
2. "Developers are motivated, technically competent, and capable of good judgement"
   - Depends critically on available staff and organizational culture
3. "Changes to the design are not ruinously expensive"
   - True only if modern technology (such as reusable components, middleware, modules, object-orientation, development tools etc.) is used *in a highly competent fashion*
   - False if an "architecture breaker" occurs
     - → almost sure if you do not have a good architecture to begin with
4. "Adjustment is easier if everything that need not be in writing, is not done in writing"
   - Possible only if there is a tightly coupled, fairly stable, and preferably co-located team
   - Not possible in a process-heavy organization:

A very heavy-weight process

DOGBERT, I NEED YOU TO FACILITATE SOME MEETINGS.
WHAT KIND OF MEETINGS?
S.Adams E-mail: SCOTTADAMS@AOL.COM
WE'RE CREATING A PROCESS TO FIX OUR PRODUCT DEVELOPMENT PROCESS. BUT FIRST WE'RE HAVING SOME PREPLANNING MEET-INGS...
2/29/96 © 1996 United Feature Syndicate, Inc.(NYC)
...TO DECIDE ON A PROJECT NAME.
HOW ABOUT "DEATH SPIRAL"?

More from Highsmith/Cockburn

"Agile Software Development: The Business of Innovation"

- Agile in a nutshell: "Agile methods stress two concepts:
  - the unforgiving honesty of working code and
  - the effectiveness of people working together with goodwill."
- Generative rules: "Most methodologies provide inclusive rules —all the things you could possibly do under all situations.
  - Agile methods offer generative rules—a minimum set of things you must do under all situations to generate appropriate practices for special situations."
- Feedback: "Because they are most applicable to turbulent, high-change environments, agile approaches recommend a variety of practices for constant feedback
  - on technical decisions, customer requirements, and management constraints."

More from Highsmith/Cockburn (2)

- Customer collaboration: "Using agile development methods requires close customer partnerships. […]
  - Poor customers result in poor systems."
- Humanist worldview: "What is new about agile methods is not the practices they use,
  - but their recognition of people as the primary drivers of project success […]"

SW processes and value generation

- Optimizing value-generation is not easy
  - we must find the "right" requirements
- Therefore, conventional (plan-driven) SW processes run a substantial **risk** of producing
  - a high overhead (→ increased cost),
    - because they build many functions that are not important
  - perhaps even low value
    - because they miss or distort some important requirements

But it is even worse:

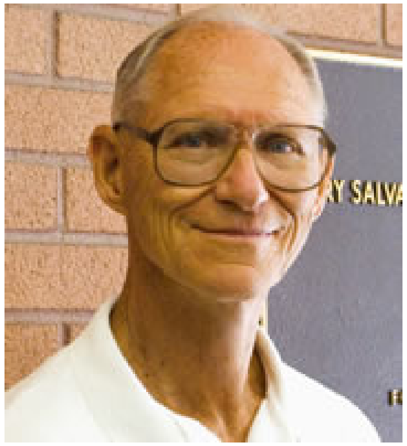
- Requirements (and hence the value proposition) change much faster and wider today than they did in the past
- Plan-driven processes cannot cope well with serious changes of a project's value proposition
  - due to their high initial investments in requirements and design

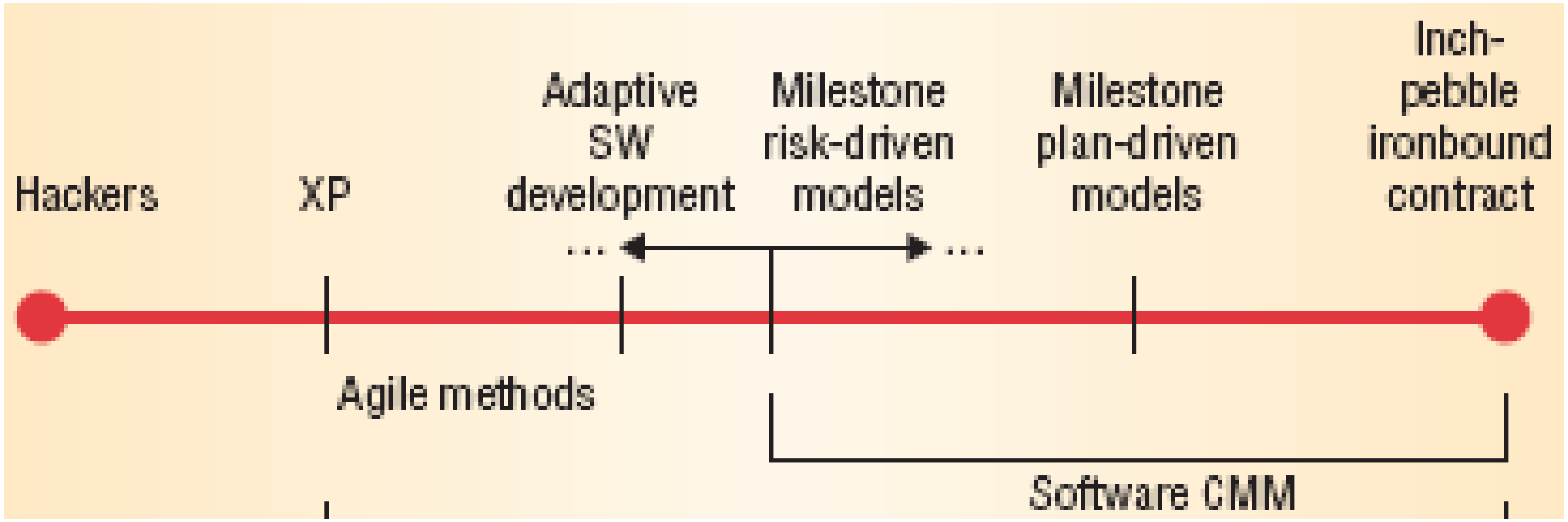
The classical view of Agile

- The most obvious difference is
  - much less planning (and in that context
  - fewer written specifications)
- Obviously, there can be too much planning or too little
  - How to think about what is a proper amount?

The planning spectrum

- Barry Boehm: "Get ready for agile methods, with care", IEEE Computer, January 2002
- The defining difference between agile and conventional methods is the amount of planning
  - There can be too much planning ("inch-pebble") as well as too little ("just hacking")




Hackers
XP
Adaptive SW development
Milestone risk-driven models
Milestone plan-driven models
Inch-pebble ironbound contract
...
...
Agile methods
Software CMM

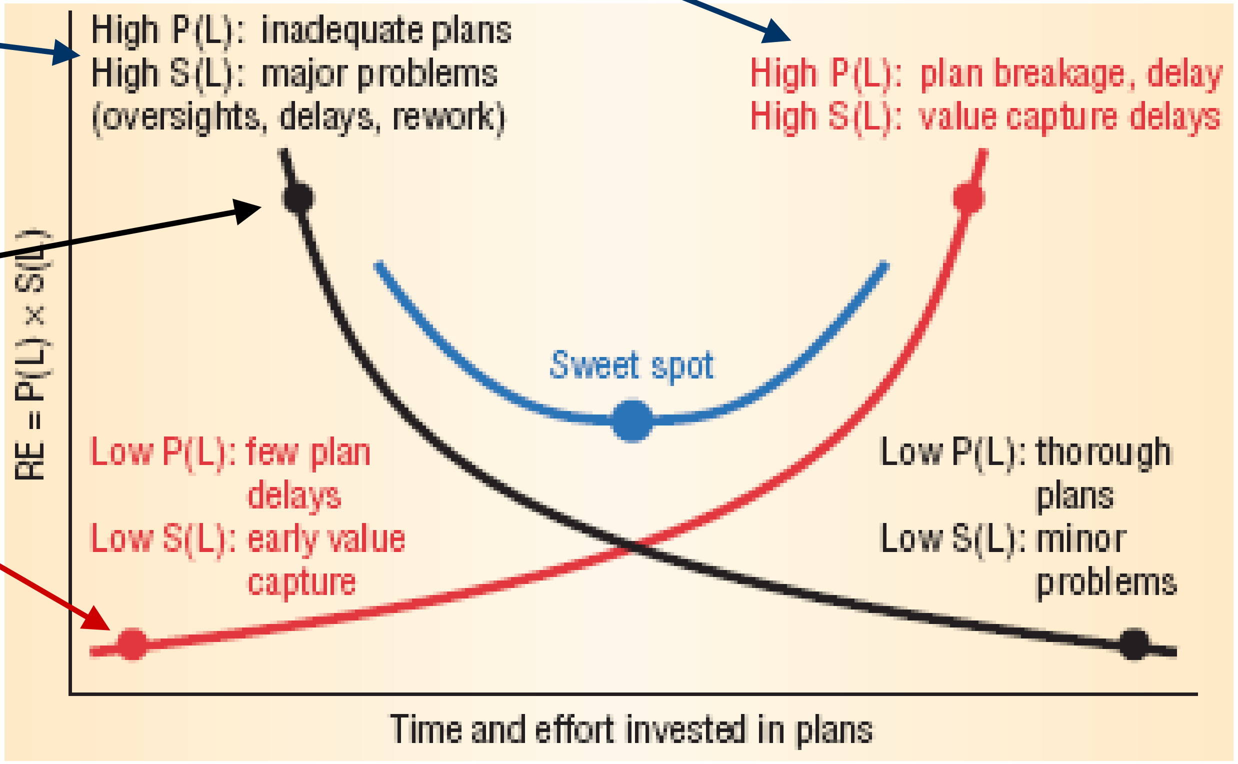
The planning spectrum (2)

- Risk exposure is the product of
  - P(L) (probability of loss) and
  - S(L) (size of loss)
- High exposure can come from too much planning or from too little

Qualitative diagram, no units!

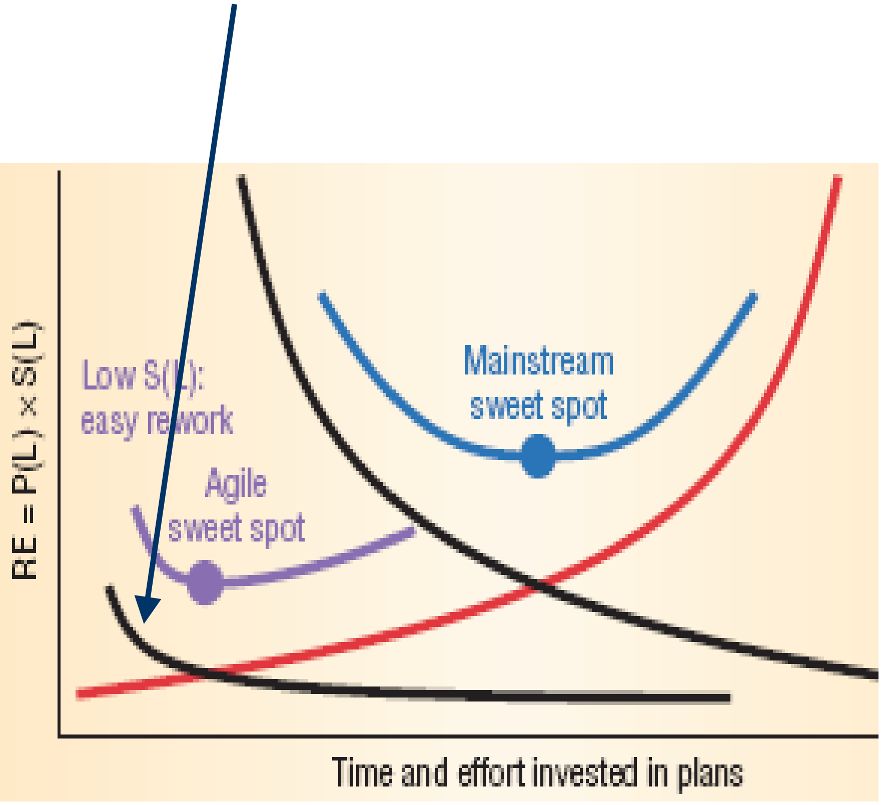
execution-related exposure

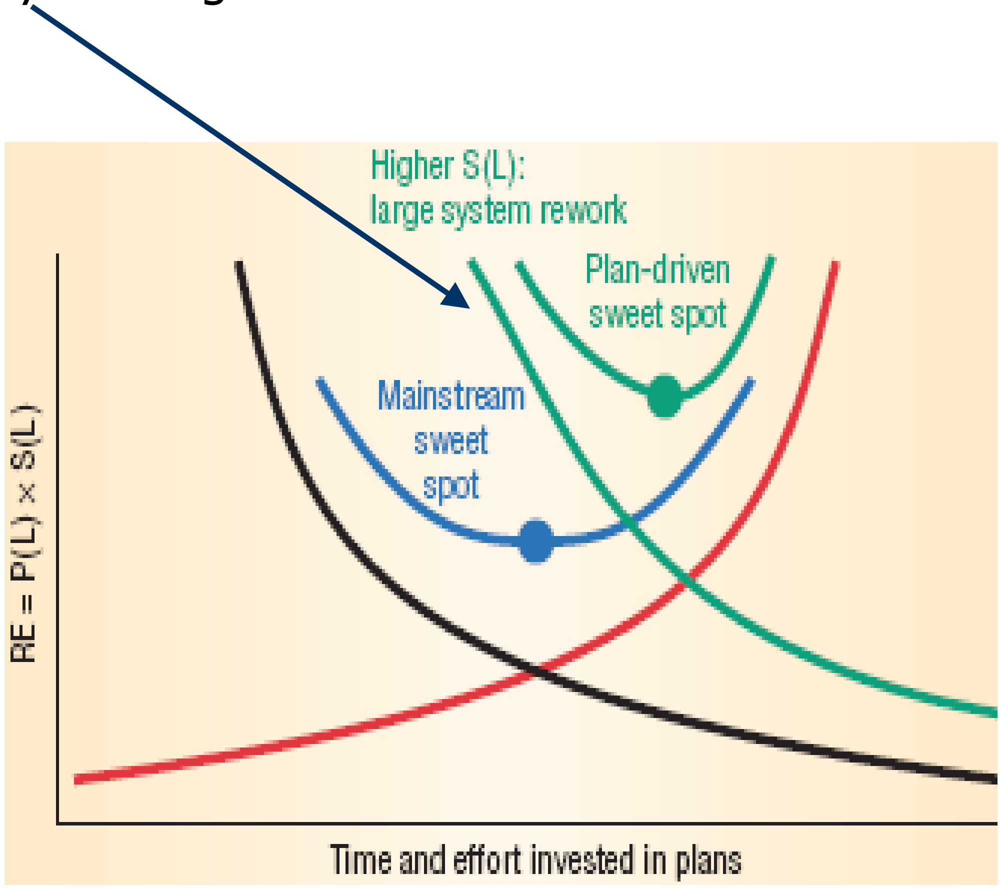
planning-related exposure



The planning spectrum (3)

- The most relevant factors are dependability requirements and project size
  - More planning is useful if they are high
  - Less if they are low





The "home ground" of agile vs. plan-driven methods

Agile methods:

- Developers:
  - agile-minded, knowledgeable, co-located, collaborative
- Requirements:
  - Largely emergent, rapid change
- Architecture:
  - Designed for current req'mts
- Refactoring:
  - Inexpensive
- Size:
  - Smaller teams and products
- Primary objective:
  - Rapid value or low requirements risk

Plan-driven methods:

- Developers:
  - plan-oriented, adequate skills, access to external knowledge
- Requirements:
  - Knowable early, largely stable
- Architecture:
  - Designed for all foreseeable requirements
- Refactoring:
  - Expensive
- Size:
  - Larger teams and products
- Primary objective:
  - High assurance of reliability

Too much planning and specification

- Specifications may remain unread if they are too detailed

YOUR REQUIREMENTS DOCUMENT IS THE BIGGEST I'VE EVER SEEN.
IT'S TOO BIG TO READ, BUT I CAN GUESS FROM ITS WEIGHT WHAT MUST BE IN THERE.
YOU KNOW IT'S A MULTI-USER, GLOBAL SYSTEM, RIGHT?
NO, I'M NOT GETTING THAT.
scottadams@aol.com
www.dilbert.com
8/9/99 © 1999 United Feature Syndicate, Inc.

https://dilbert.com/strip/1999-08-09

"Just enough" planning: The XP Planning Game

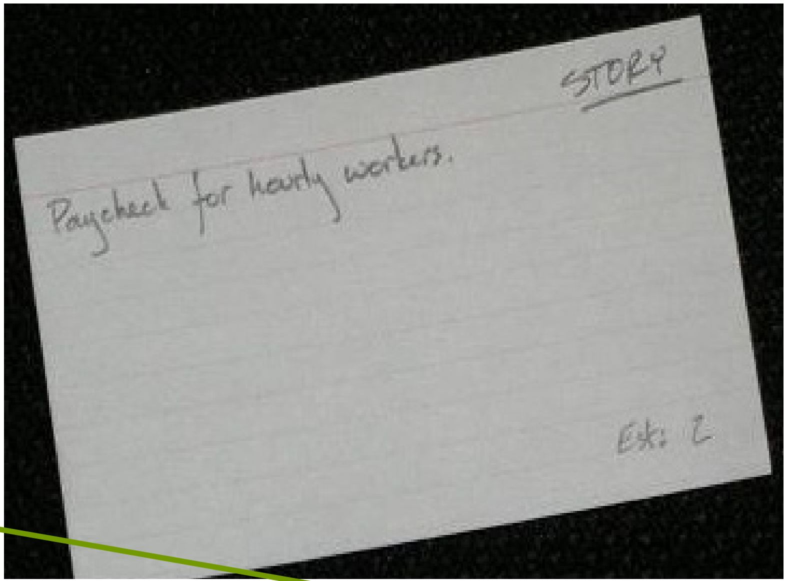
- Extreme Programming (XP) is one of several concrete agile development methods
  - It consists of a number of specific practices (see next lecture)
  - One of these is called "Planning Game" and describes the project planning method
- XP project planning occurs on three granularity levels

Release	~2-4 project months
Iteration	~1-2 project weeks
Task	~0.5-3 person days

- Only the *current* Release, Iteration, and set of Tasks are planned in some detail
  - all future Releases and Iterations (even within the current release) are at most sketched, perhaps not even that

XP Planning Game: 1. Release planning

- 1. Customer enumerates (rough) requirements
  - Each is written on a *Story Card*
  - Cards are collected
- 2. Customer prioritizes stories into E (essential), V (valuable), and N (nice-to-have)
- 3. Developers query Customer to obtain sufficient detail about stories to understand their content and purpose
  - "conversation"
- 4. Developers estimate development cost for each story
  - and categorize their estimates into R (reliable), A (approximate), and U (unknown)
- 5. Customer selects stories for next release, prefering E and V
  - Remaining stories are to be realized in subsequent releases



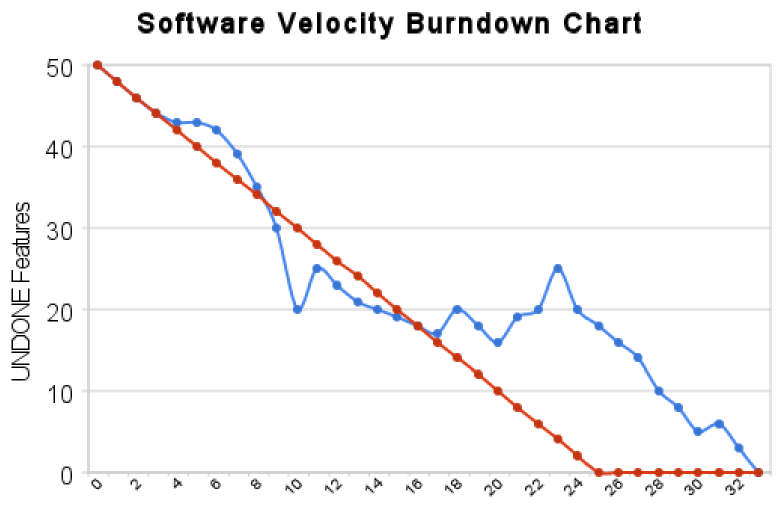
Value Risk!

XP Planning Game:
2. Iteration and task planning

- Before each iteration, the customer can select stories for it
  - Priorities may have shifted
  - Estimation may now be more precise
  - Customer is even allowed to bring in new requirements
    - but must drop (for this release) others of the same weight
- Customer defines acceptance tests for selected stories
  - "confirmation"
- and clarifies remaining details about the stories
  - "conversation"

- Developers turn the set of stories (requirements) into a set of tasks (design and implementation work)
  - A developer adopts a task and then
    must personally estimate its size

XP Planning Game: 3. Plan tracking and replanning

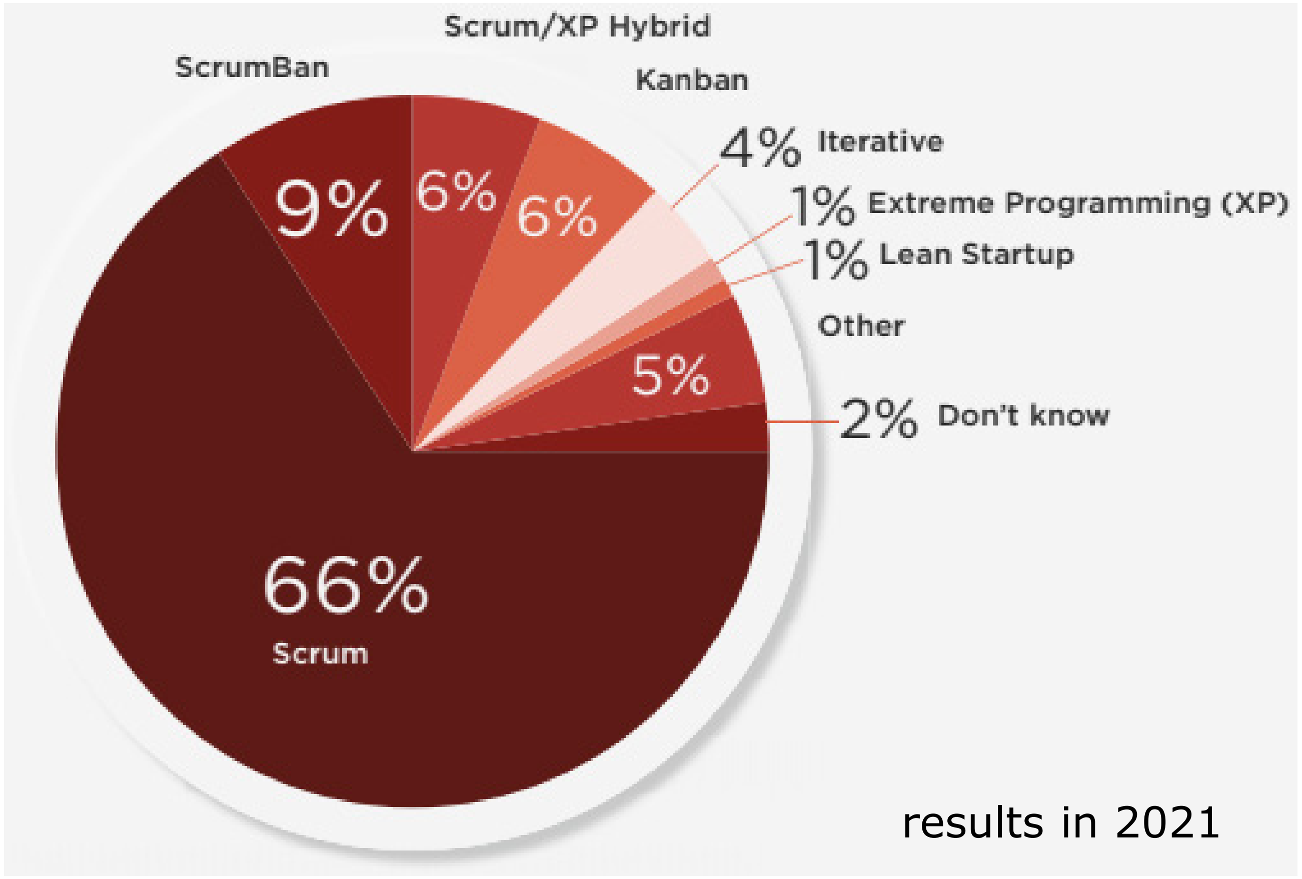


- Tracking the release-level plan
  - after each iteration, compare expected and actual progress → possibly modify release content
- Tracking the iteration-level plan
  - developers report completed tasks daily → possibly modify iteration content

- Estimation is always done in terms of "ideal development time"
  - i.e., programming only, without interruptions or additional tasks.
  - After each iteration, a "load factor" is computed for each developer, relating ideal time to elapsed time
  - and is used during the next iteration planning
- Planning focus is always on stories
  - because they represent customer value

Which processes used how often? State of Agile survey [digitalai21]

- yearly survey, ~4000 respondents
  - ~40% North Am., ~40% Europe
  - ~50% team members, ~50% managers, consultants etc.



Summary

- Agile approaches expect change and attempt to accommodate it as much as possible
- They attempt to avoid process elements that make change expensive
  - and replace them by something that makes change cheaper

- Agile approaches are most suitable when
  - team sizes are small,
  - dependability requirements are modest, and
  - requirements change is common

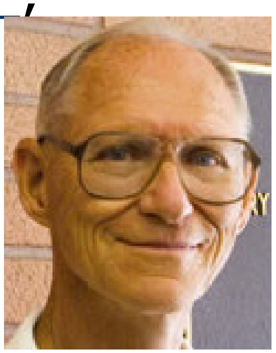
- Agile processes apply a very different planning style
  - coarser
    - postponing much detail-clarification to implementation time
  - much shorter-term
  - priority is on ensuring value, not finishing on time

Thank you!

next: slides on balancing agile vs. plan-driven

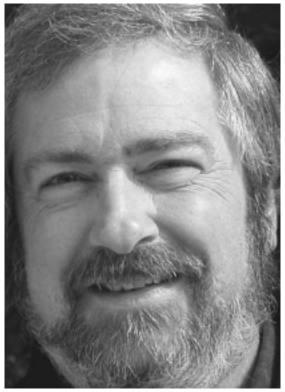
Using Risk for Balancing Agile vs. Plan-Driven

- Barry Boehm, Richard Turner: *"Using Risk to Balance Agile and Plan-Driven Methods"*, IEEE Computer, June 2003


B. Boehm

Basic idea:

- Agile methods are susceptible to different risks than plan-driven methods
- Analyzing risk types and strengths helps deciding which method (or balance of methods) to use


R. Turner

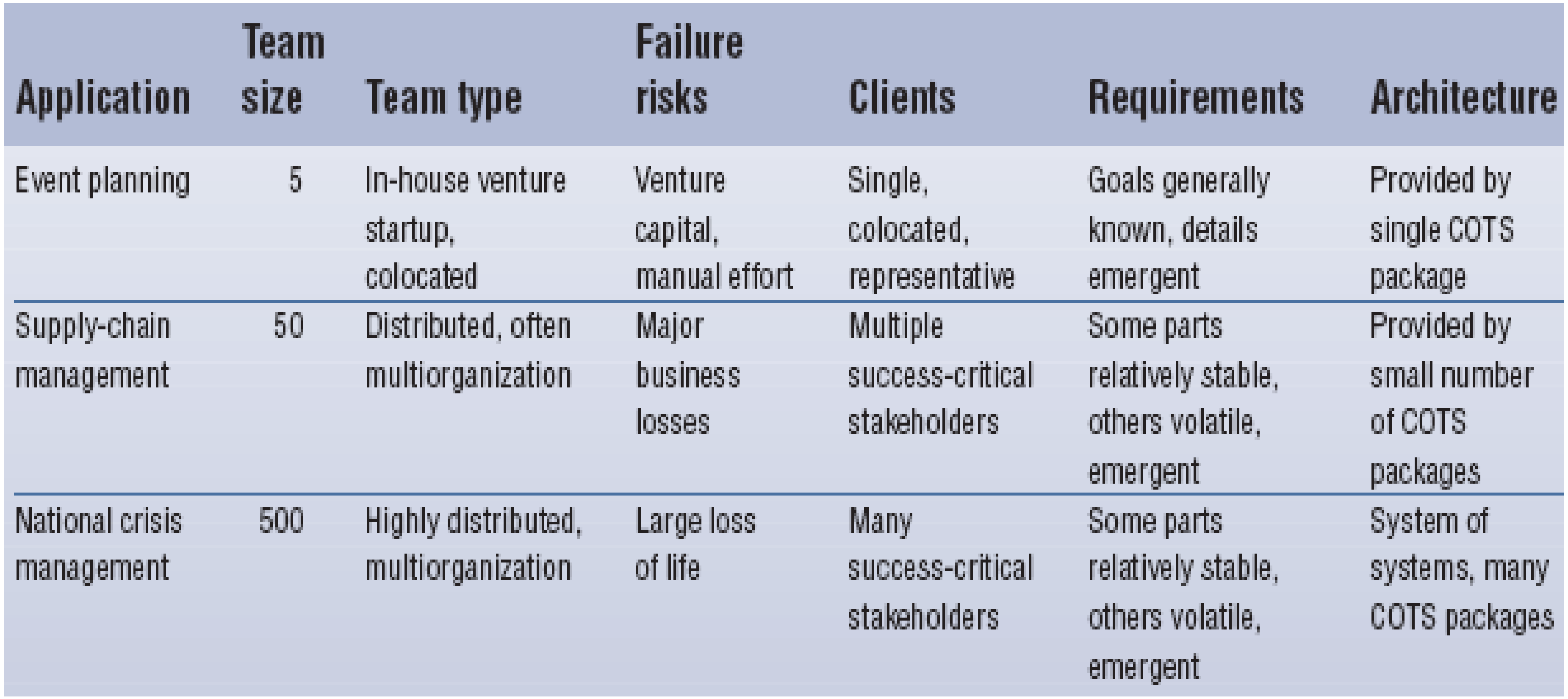
Approach:

- Classify risks into

• environmental risks	(unavoidable)
• agility-oriented risks	(reducible by planning)
• plan-driven risks	(reducible by agility)

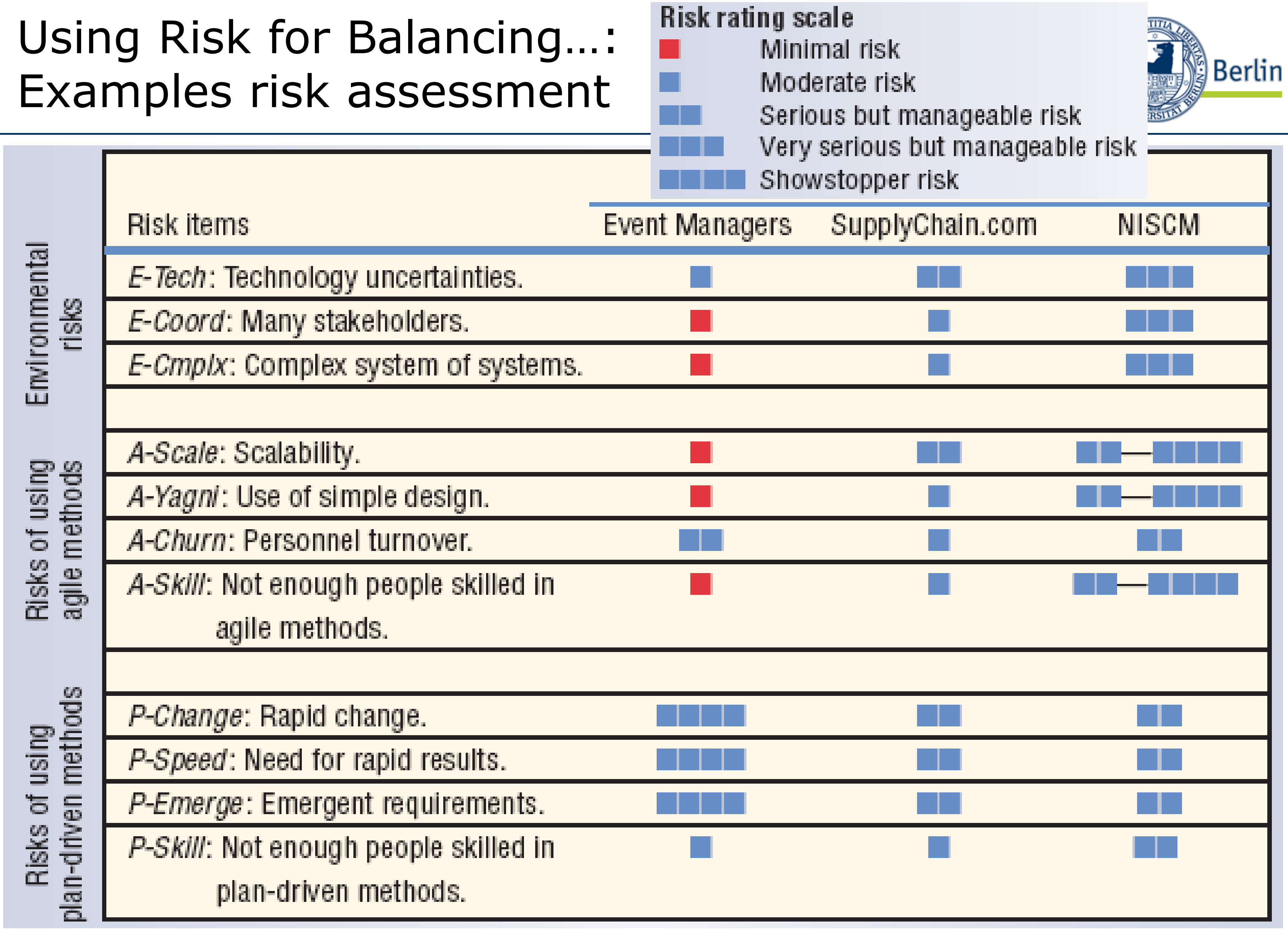
Using Risk for Balancing…: Examples

- Characteristics of 3 imaginary example projects/systems:



Application	Team size	Team type	Failure risks	Clients	Requirements	Architecture
Event planning	5	In-house venture startup, colocated	Venture capital, manual effort	Single, colocated, representative	Goals generally known, details emergent	Provided by single COTS package
Supply-chain management	50	Distributed, often multiorganization	Major business losses	Multiple success-critical stakeholders	Some parts relatively stable, others volatile, emergent	Provided by small number of COTS packages
National crisis management	500	Highly distributed, multiorganization	Large loss of life	Many success-critical stakeholders	Some parts relatively stable, others volatile, emergent	System of systems, many COTS packages

Using Risk for Balancing…: Examples risk assessment



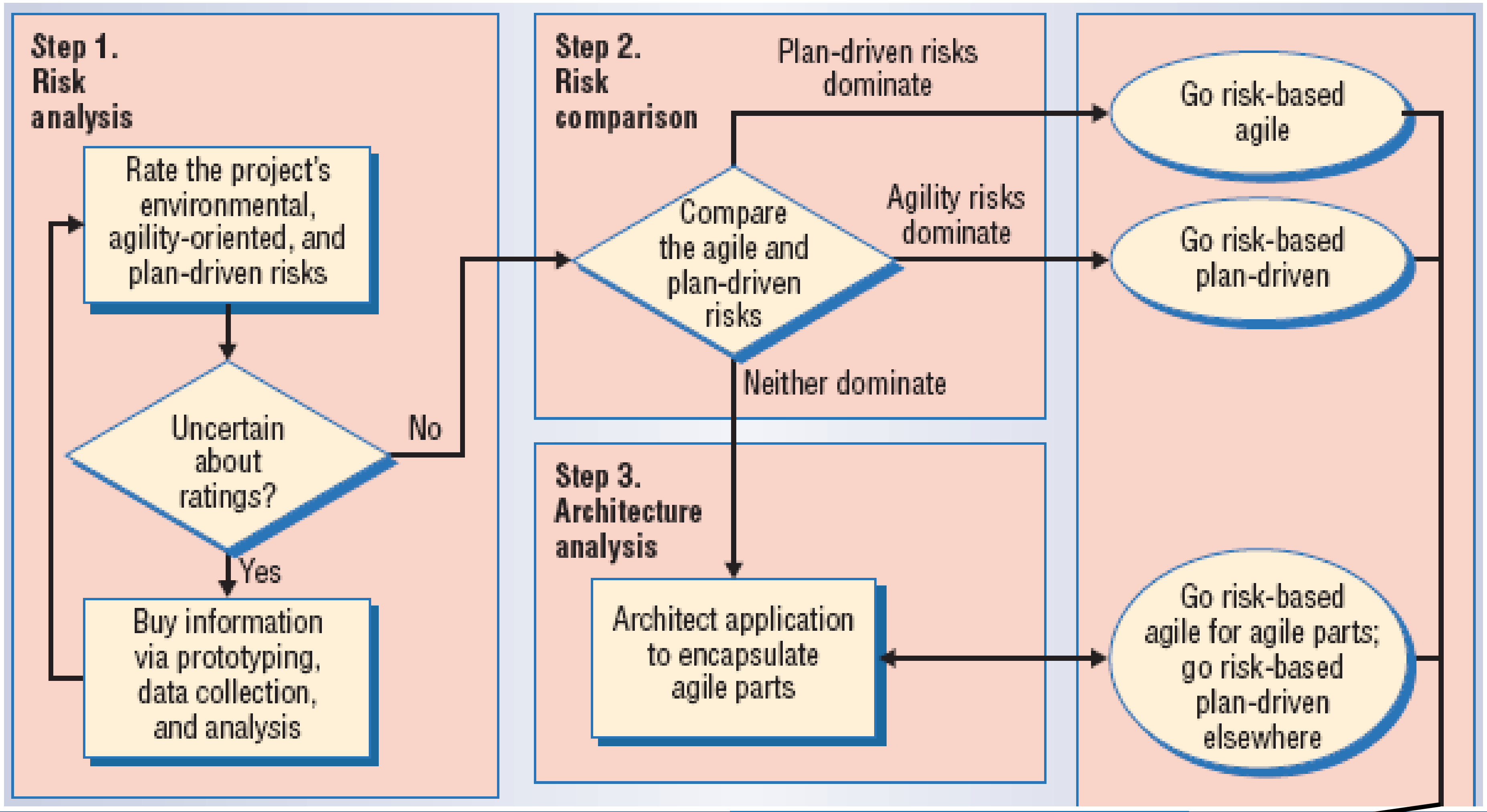
Risk rating scale

🟥	Minimal risk
🟦	Moderate risk
🟦🟦	Serious but manageable risk
🟦🟦🟦	Very serious but manageable risk
🟦🟦🟦🟦	Showstopper risk

	Risk items	Event Managers	SupplyChain.com	NISCM
Environmental risks	*E-Tech*: Technology uncertainties.	🟦	🟦🟦	🟦🟦🟦
	E-Coord: Many stakeholders.	🟥	🟦	🟦🟦🟦
	E-Cmplx: Complex system of systems.	🟥	🟦	🟦🟦🟦
Risks of using agile methods	*A-Scale*: Scalability.	🟥	🟦🟦	🟦🟦—🟦🟦🟦🟦
	A-Yagni: Use of simple design.	🟥	🟦	🟦🟦—🟦🟦🟦🟦
	A-Churn: Personnel turnover.	🟦🟦	🟦	🟦🟦
	A-Skill: Not enough people skilled in agile methods.	🟥	🟦	🟦🟦—🟦🟦🟦🟦
Risks of using plan-driven methods	*P-Change*: Rapid change.	🟦🟦🟦🟦	🟦🟦	🟦🟦
	P-Speed: Need for rapid results.	🟦🟦🟦🟦	🟦🟦	🟦🟦
	P-Emerge: Emergent requirements.	🟦🟦🟦🟦	🟦🟦	🟦🟦
	P-Skill: Not enough people skilled in plan-driven methods.	🟦	🟦	🟦🟦

Using Risk for Balancing…: Decision schema

- High risk uncertainty → "buy" information


Step 1. Risk analysis
Rate the project's environmental, agility-oriented, and plan-driven risks
Uncertain about ratings?
No
Yes
Buy information via prototyping, data collection, and analysis
Step 2. Risk comparison
Compare the agile and plan-driven risks
Plan-driven risks dominate
Agility risks dominate
Neither dominate
Go risk-based agile
Go risk-based plan-driven
Step 3. Architecture analysis
Architect application to encapsulate agile parts
Go risk-based agile for agile parts; go risk-based plan-driven elsewhere

now tailor, execute, and monitor

Using Risk for Balancing…: Levels of development staff

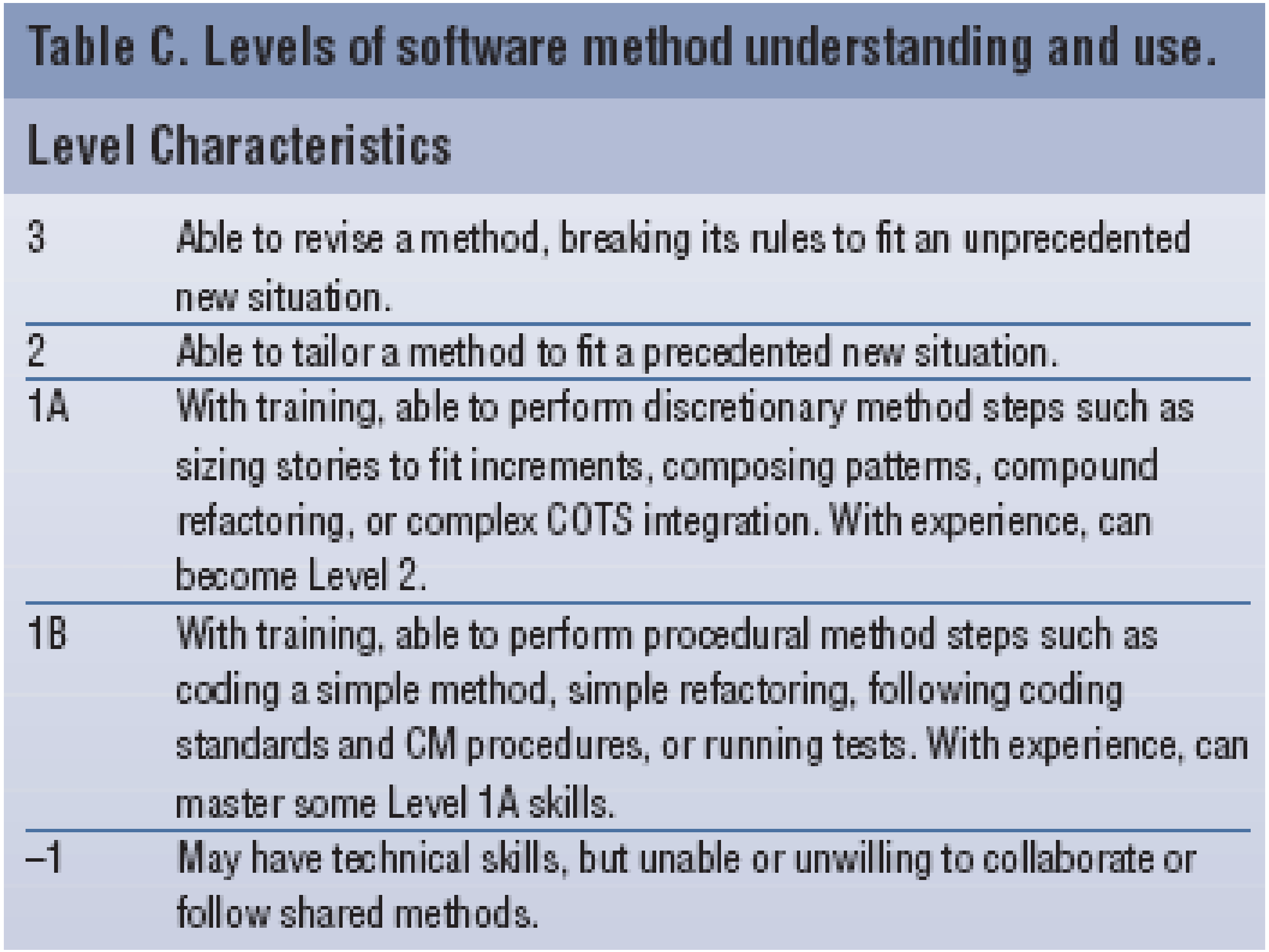


Table C. Levels of software method understanding and use.

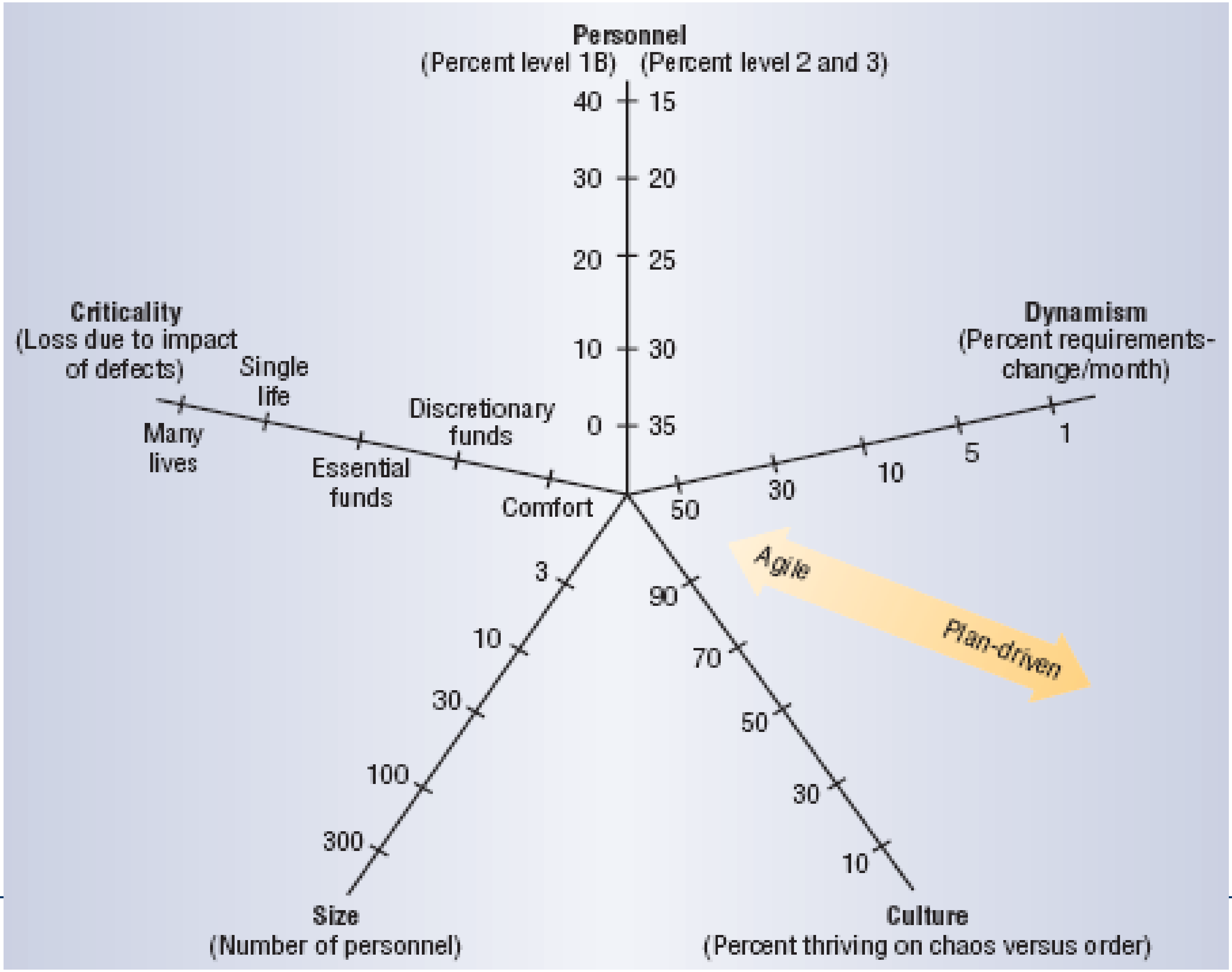
Level	Characteristics
3	Able to revise a method, breaking its rules to fit an unprecedented new situation.
2	Able to tailor a method to fit a precedented new situation.
1A	With training, able to perform discretionary method steps such as sizing stories to fit increments, composing patterns, compound refactoring, or complex COTS integration. With experience, can become Level 2.
1B	With training, able to perform procedural method steps such as coding a simple method, simple refactoring, following coding standards and CM procedures, or running tests. With experience, can master some Level 1A skills.
–1	May have technical skills, but unable or unwilling to collaborate or follow shared methods.

A 5:1 ratio of Level 1A to Level 2 staff can be OK for agile methods, but 1B get in the way.

Plan-driven can function with many 1B.

-1 are always a problem.

Using Risk for Balancing…: Risks and risk levels


Personnel
(Percent level 1B)
(Percent level 2 and 3)
Criticality
(Loss due to impact of defects)
Many lives
Single life
Essential funds
Discretionary funds
Comfort
Dynamism
(Percent requirements-change/month)
Size
(Number of personnel)
Culture
(Percent thriving on chaos versus order)
Agile
Plan-driven