Course "Softwareprozesse"

Agile Methods

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik

- The Waterfall myth
- Goals and priorities in SW development
- Principles of agile approaches
  - Misunderstandings about them
- Assumptions of agile methods
- Balancing risk:
  - Too much vs. too little planning
    - The XP planning game
  - Other
- CMMI process areas in agile methods

Learning objectives

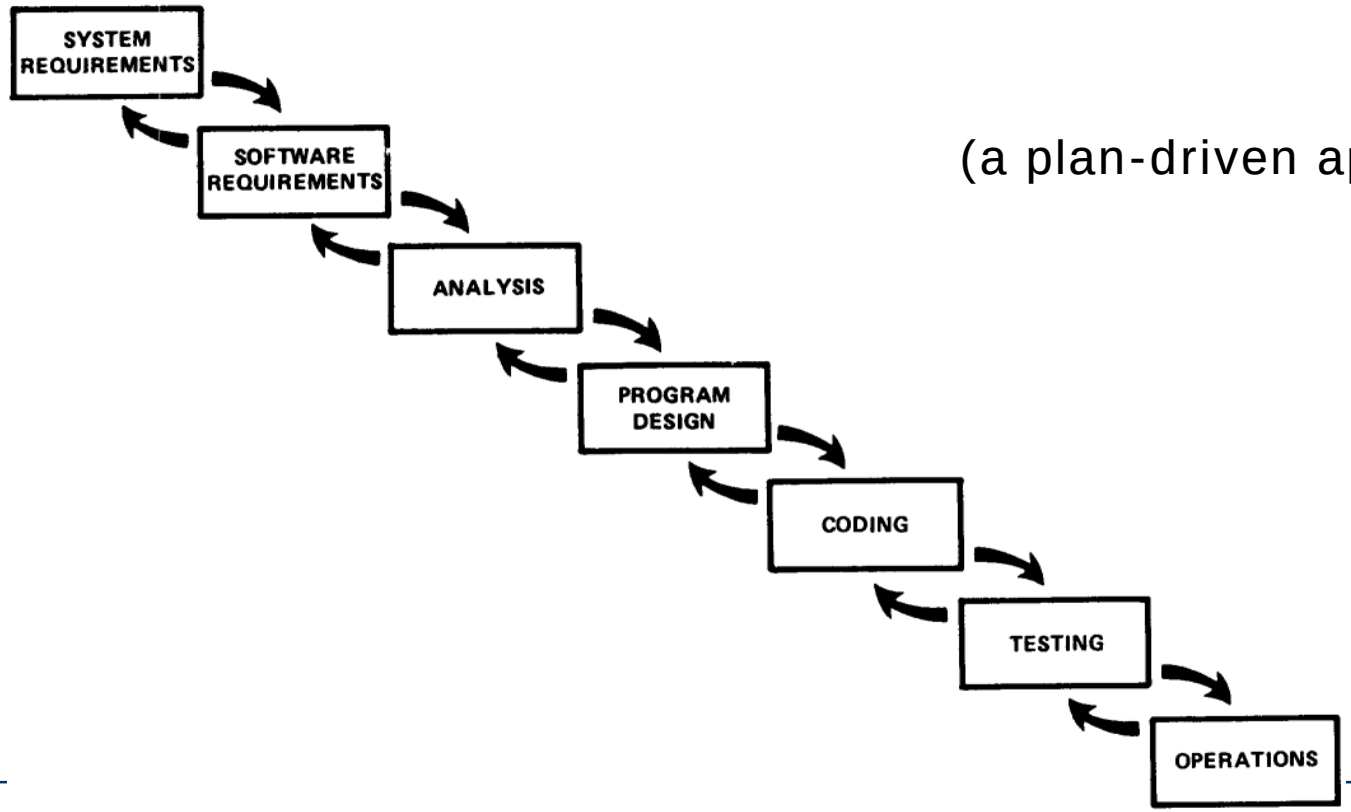
- Understand the basic ideas of agile methods
- Contrast them to "conventional" approaches
- Understand the best-fit regions for application of agile methods
- Understand risk as a driver for agility decisions

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.

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 are usually 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
- He illustrates the complete proposal as follows:

The Waterfall myth (3)

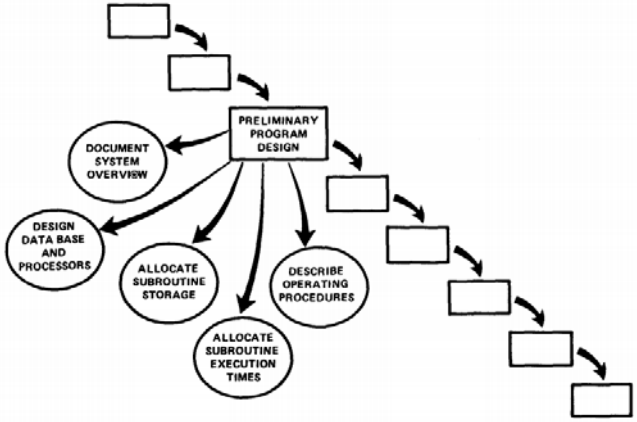


Figure 5. Step 1: Insure that a preliminary program design is complete before analysis begins.

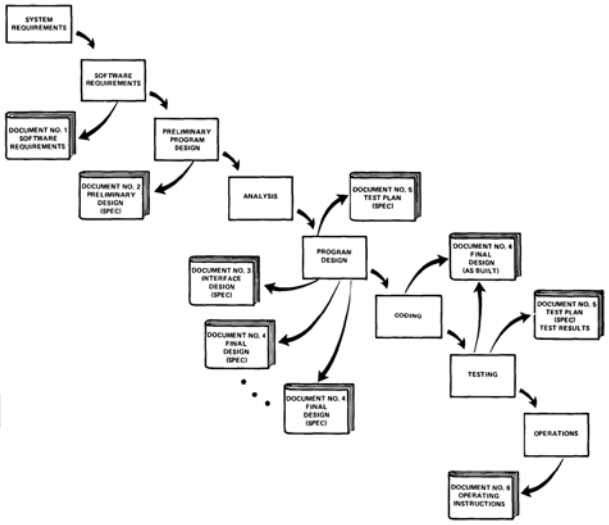


Figure 6. Step 2: Insure that documentation is current and complete – at least six uniquely different documents are required.

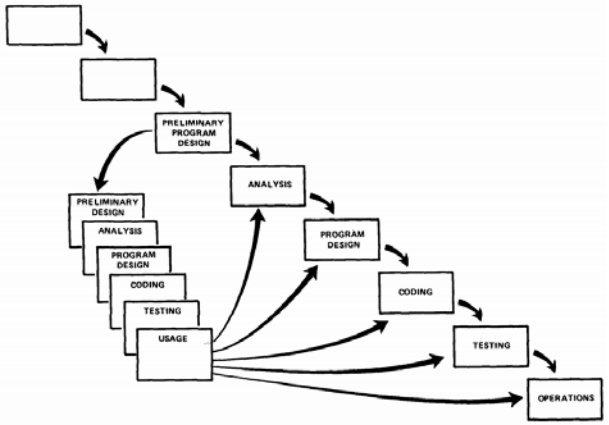


Figure 7. Step 3: Attempt to do the job twice – the first result provides an early simulation of the final product.

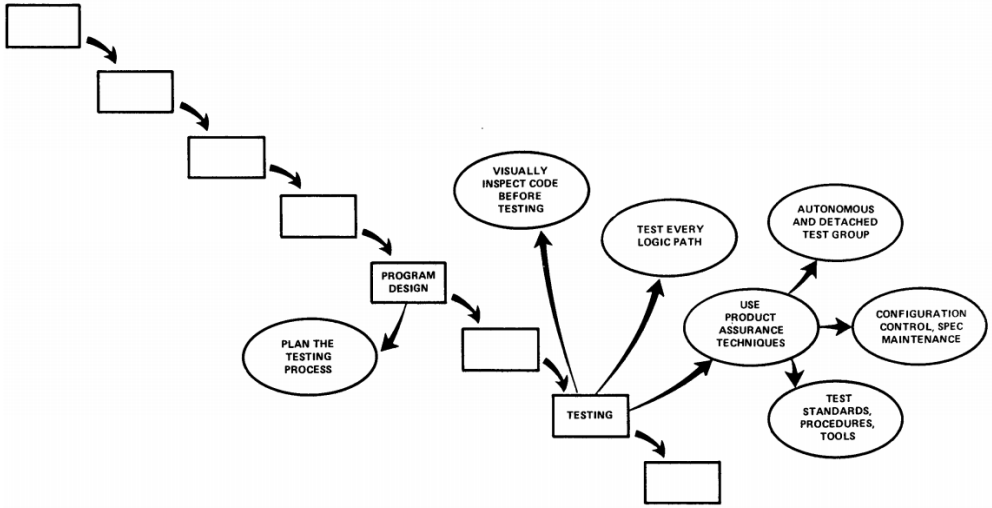


Figure 8. Step 4: Plan, control, and monitor computer program testing.

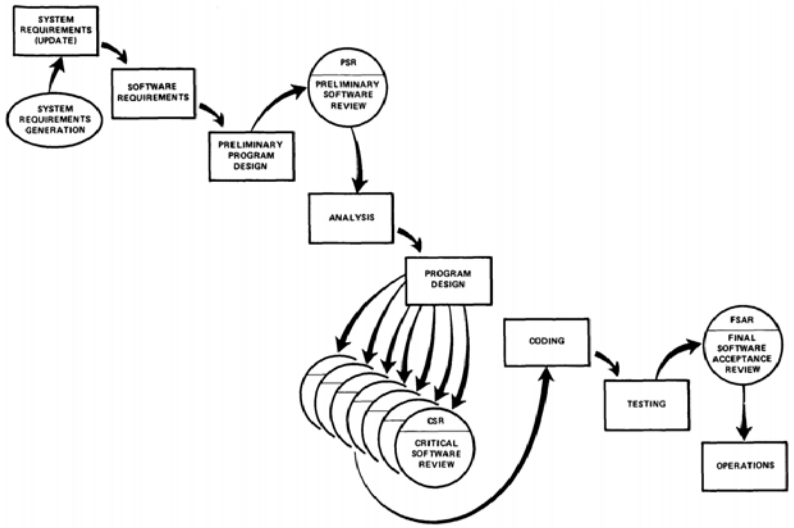
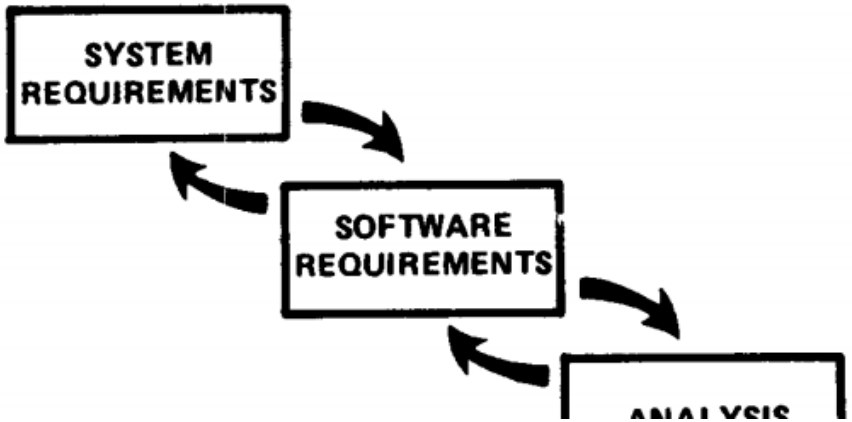


Figure 9. Step 5: Involve the customer – the involvement should be formal, in-depth, and continuing.

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)



SW processes and value generation

- As we saw in the lecture on SW engineering economics, optimizing value-generation is not easy
- 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

Purpose of software processes

One possible view (but more facets exist):

- **Plan-driven:**
  1. Anticipate development as much as possible in order to eliminate the cost of changes
  2. Generate high quality at low cost
- **Agile:**
  1. Find ways to *reduce* the cost of change because eliminating change is impossible
  2. Generate high value at low cost and high speed

- Jim Highsmith, Alistair Cockburn: "*Agile Software Development: The Business of Innovation*", IEEE Computer, September 2001
  - Very good introduction into the agile way of thinking

Agile is about priorities

Highsmith and Cockburn:

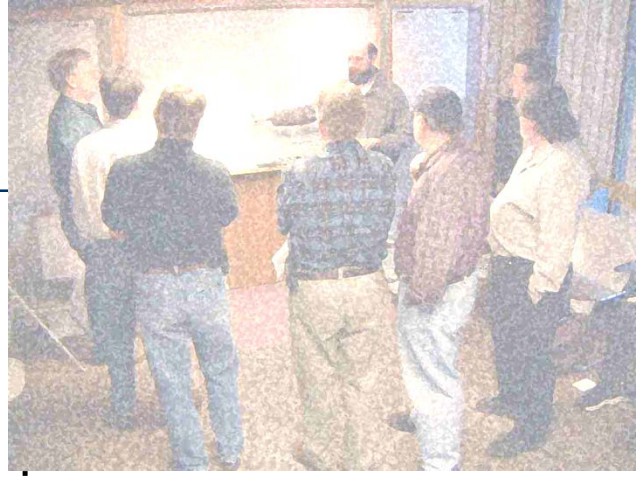
- *"Processes, tools, documentation, contracts, and plans are useful.*
  But when push comes to shove — and it usually does — something must give, and we need to be clear about what stays and what gives."
  - (Agile methods were initially called "lightweight" methods, because they attempted to get rid of dispensable process elements)

- These priorities were spelled out in the oft-misunderstood Agile Manifesto
  - published by a group of 17 high-profile members of the agile process community
  - all of them are practitioners who really know what they are talking about

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."

Agile Manifesto: Principles

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. Working software is the primary measure of progress.
   - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
3. Welcome changing requirements, even late in development.
   - Agile processes harness change for the customer's competitive advantage.
4. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
   - In particular, business people and developers must work together daily throughout the project.

Agile Manifesto: Principles (2)

5. Agile processes promote sustainable development.
   - The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
6. Build projects around motivated individuals.
   - Give them the environment and support they need, and trust them to get the job done.
   - Continuous attention to technical excellence and good design enhances agility.
   - The best architectures, requirements, and designs emerge from self-organizing teams.
   - At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
7. Simplicity is essential.
   - Simplicity is the art of maximizing the amount of work not done.

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://www.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.

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
   - User documentation may, too (Depends!)
   - 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 that of cheap design changes

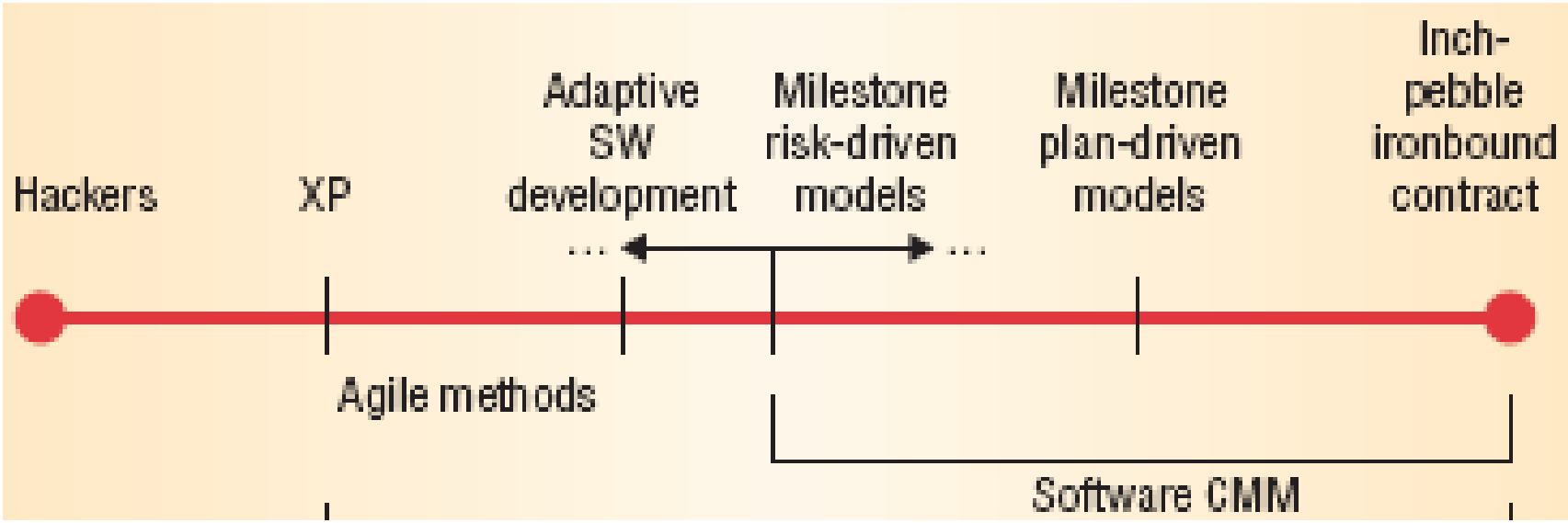
When are the assumptions true?

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

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")



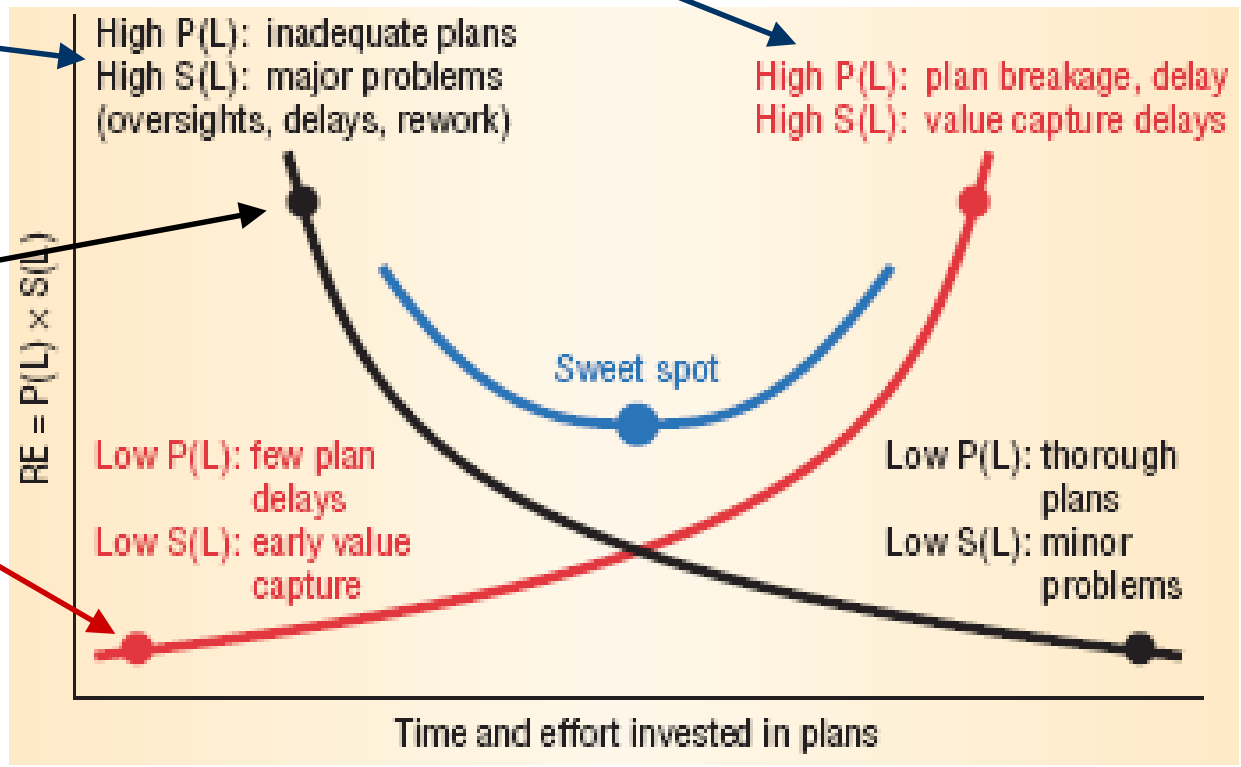
The planning spectrum (2)

- 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!

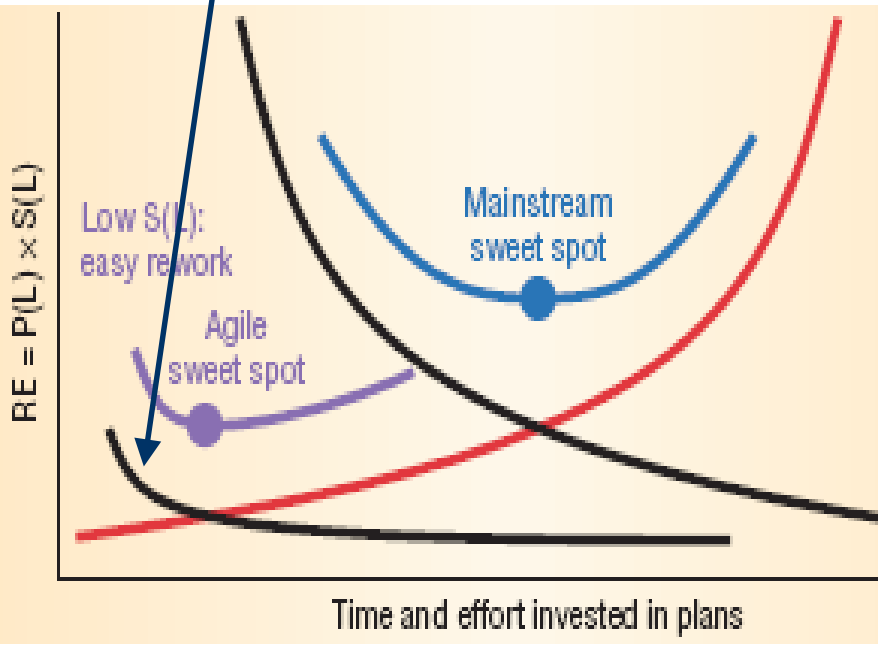
execution-related exposure

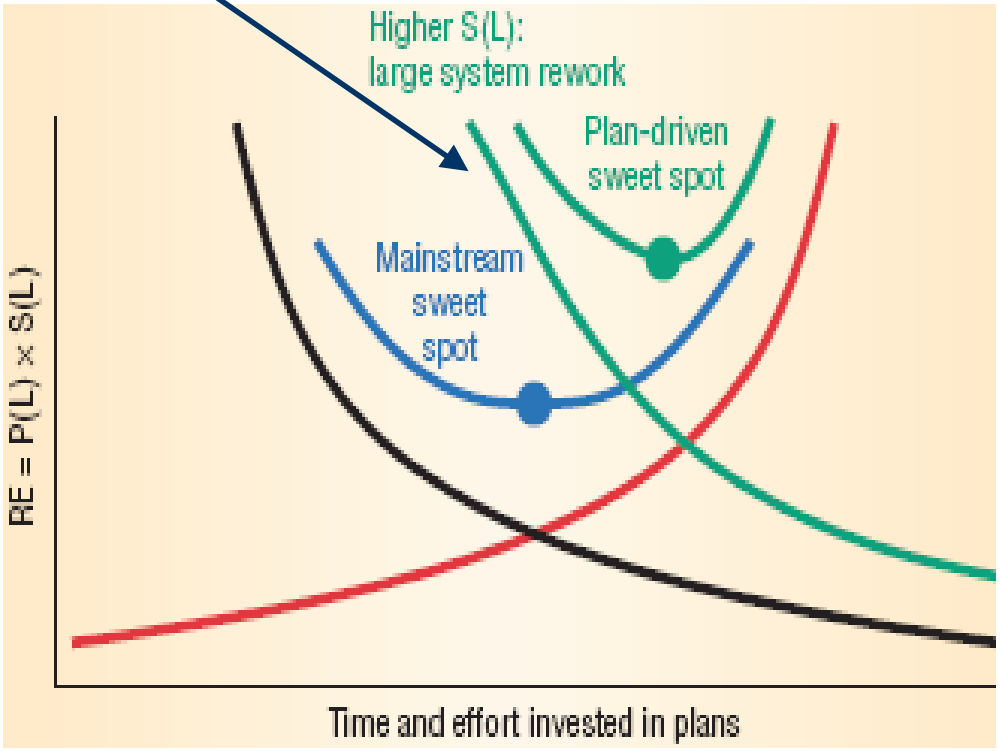
planning-related exposure



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

Freie Universität Berlin

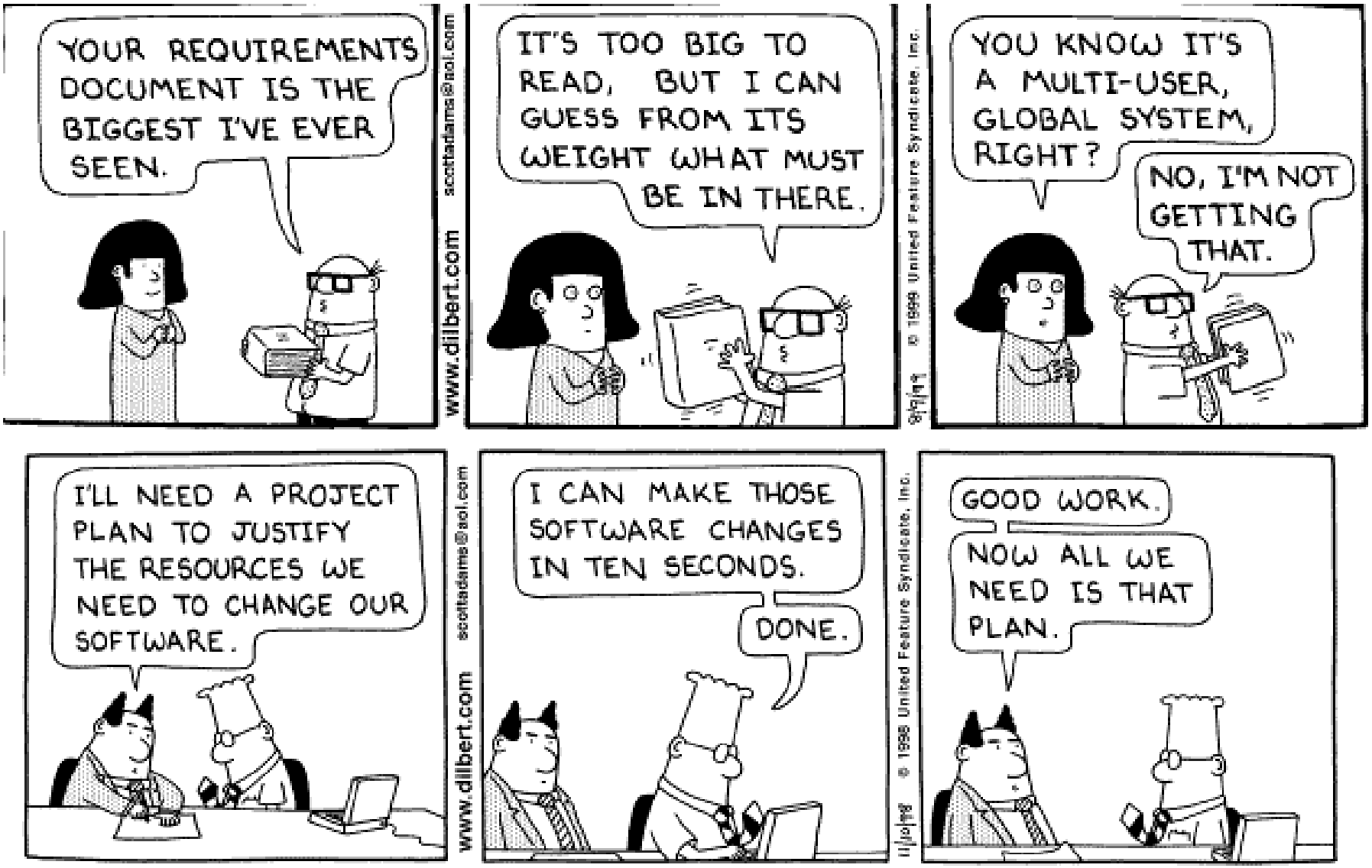
Agile methods:

- 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 current and foreseeable requirements
- Refactoring:
  - Expensive
- Size:
  - Larger teams and products
- Primary objective:
  - High assurance of reliability

Too much planning



"Just enough" planning: The XP Planning Game

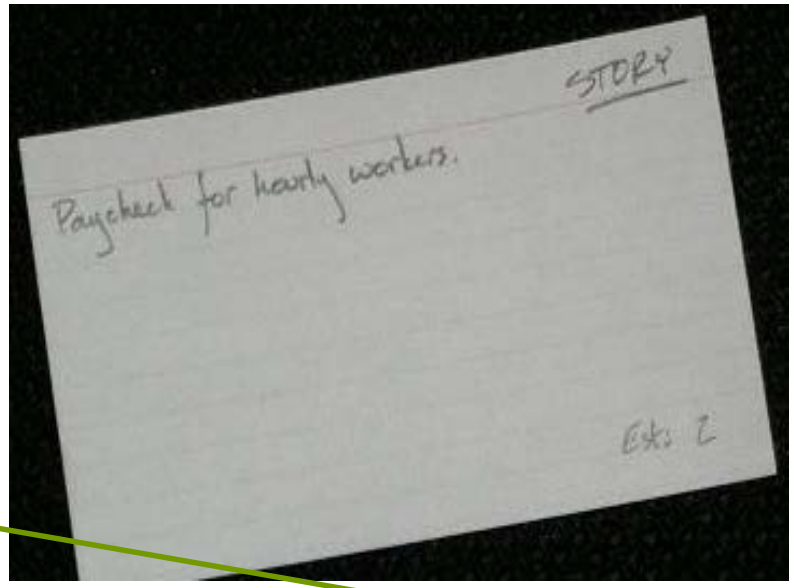
- Extreme Programming (XP) is one of several concrete agile development methods
  - It consists of a number of specific practices (see next lecture)
  - One of these is called "Planning Game" and describes the project planning method

- 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



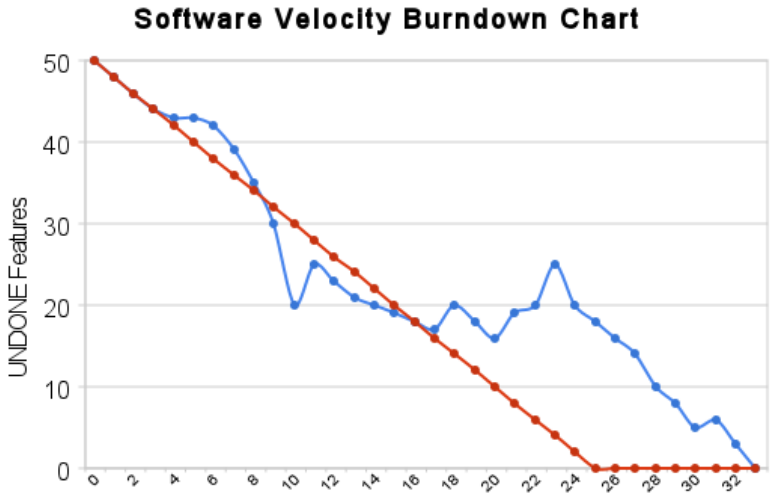
Value Risk!

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

Other elements of agile processes

Besides lean planning, there are other typical behaviors that help with agility:

- **Documentation:**
  - Whenever you can, rely on oral communication instead of written information
    - Rationale 1: Talking is so much cheaper than writing that you can even tolerate some redundance; in the presence of change, writing costs can probably never pay off
    - Rationale 2: Oral communciation is more flexible, understanding is easier to obtain
  - If you have to use writing for clarity, consider throwing it away as soon as understanding has been achieved
    - Rationale: In the presence of change, it is too costly to keep the documentation up-to-date; out-of-date documents are dangerous.
- **Automation:**
  - Use automation wherever possible to remove barriers against change (in particular: build process, testing, perhaps deployment)

Other elements of agile processes (2)

- **Meetings** and communication:
  - Keep meetings with multiple participants as short as possible
    - Rationale: Multi-person meetings are inefficient [Why?]
  - Rely a lot on one-to-one discussions with changing partners
    - Rationale: Can replace many meetings; avoids meetings' tendency to get lengthy; improves motivation and initiative
  - Maximize the opportunity for informal communication
    - Rationale: Reduces the need for formal meetings as risk-reduction mechanisms
- Early and intensive **feedback**:
  - Reduce the uncertainty about the quality of requirements (and implementation) by frequent releases
  - Reduce the uncertainty about the consistency of design and implementation by automated builds and tests
- **Simplicity**:
  - Avoid all complexity that is not very clearly warranted

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

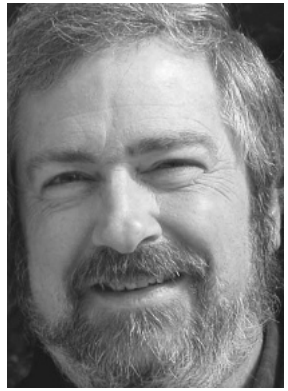
B. Boehm

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

Approach:

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



R. Turner

Using Risk for Balancing…: Examples

Freie Universität Berlin

- 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

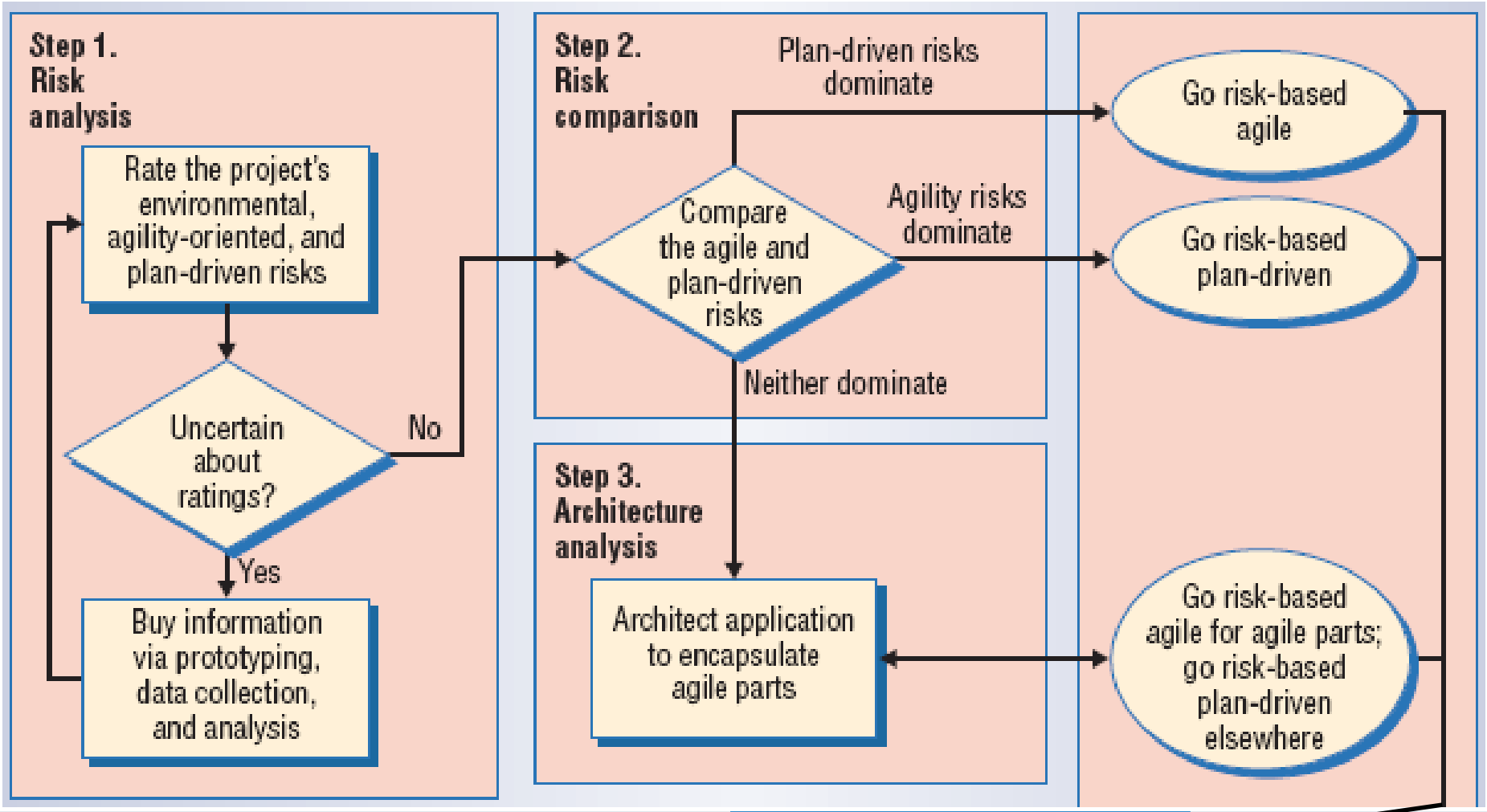
Risk rating scale

- 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



now tailor, execute, and monitor

Using Risk for Balancing…: Levels of development staff

Freie Universität Berlin

Table C. Levels of software method understanding and use.

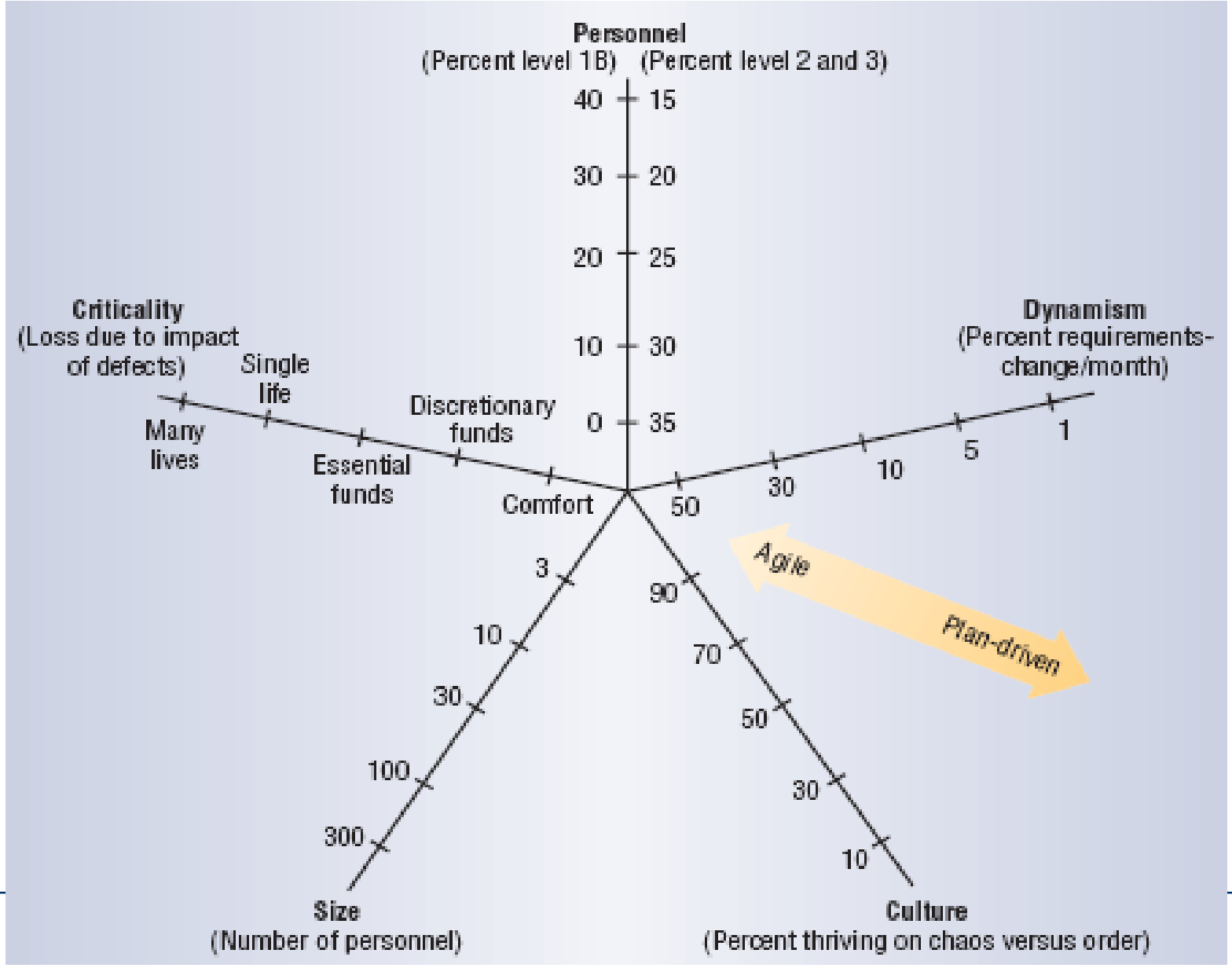
Level	Characteristics
3	Able to revise a method, breaking its rules to fit an unprecedented new situation.
2	Able to tailor a method to fit a precedented new situation.
1A	With training, able to perform discretionary method steps such as sizing stories to fit increments, composing patterns, compound 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



CMMI process areas in agile methods

Freie Universität Berlin

- Level 2: Managed
  - + Requirements Mgmt
  - **o Project Planning**
  - + Project Monitoring&Control
  - - Supplier Agreement Mgmt
  - o Measurement and Analysis
  - o Process and Product Quality Assurance
  - + Configuration Management
- Level 3: Defined
  - + Req's. Development
  - + Technical Solution
  - o Product Integration
  - - Verification
  - **+++ Validation**
  - - Organizational Process Focus
  - - Organ'l Process Definition
  - - Organizational Training
  - o Integrated Project Mgmt.
  - o Risk Management
  - o Decision Analysis and Resolution
- Level 4: Quantitatively Manag'd
  - - Organizational Process Performance
  - o Quantitative Project Mgmt
- Level 5: Optimizing
  - o Organizational Performance Management
  - o Causal Analysis and Resolution

+ usually available
o avail. in reduced form
- usually mostly absent

Further resources

- http://www.agilealliance.com
  - A community portal around the agile approach.
    Has pointers to most everything about it.
  - In particular, a **guide to practices**: http://guide.agilealliance.org/subway.html
- http://alistair.cockburn.us/
  - Alistair Cockburn (one of the best writers-of-agile), Crystal
- http://www.controlchaos.com/
  - Ken Schwaber, SCRUM
- http://www.xprogramming.com
  - Ron Jeffries, XP
- http://www.cio.com/article/print/464169
  - "When Agile Projects Go Bad": A good article about common mistakes when going agile
- IEEE Computer, June 2003
  - Issue topic "Agile Methods" (7 articles about when and how)
  - http://www.computer.org

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

- A disciplined agile process can cover many of the CMMI process areas up to level 5
  - however in rather unconventional ways

Thank you!