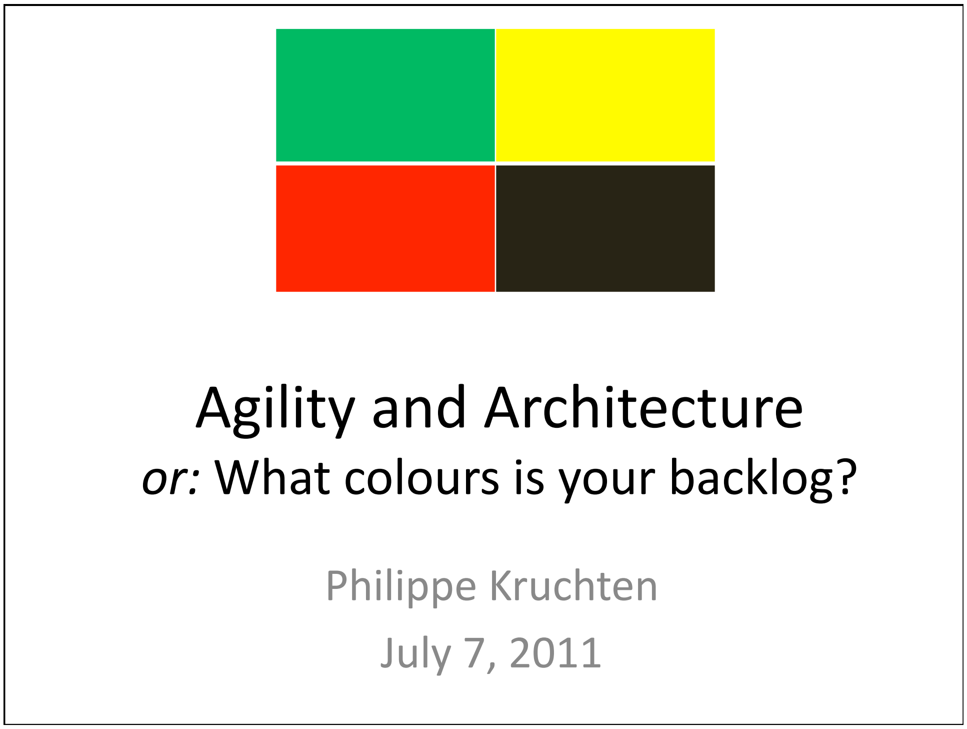
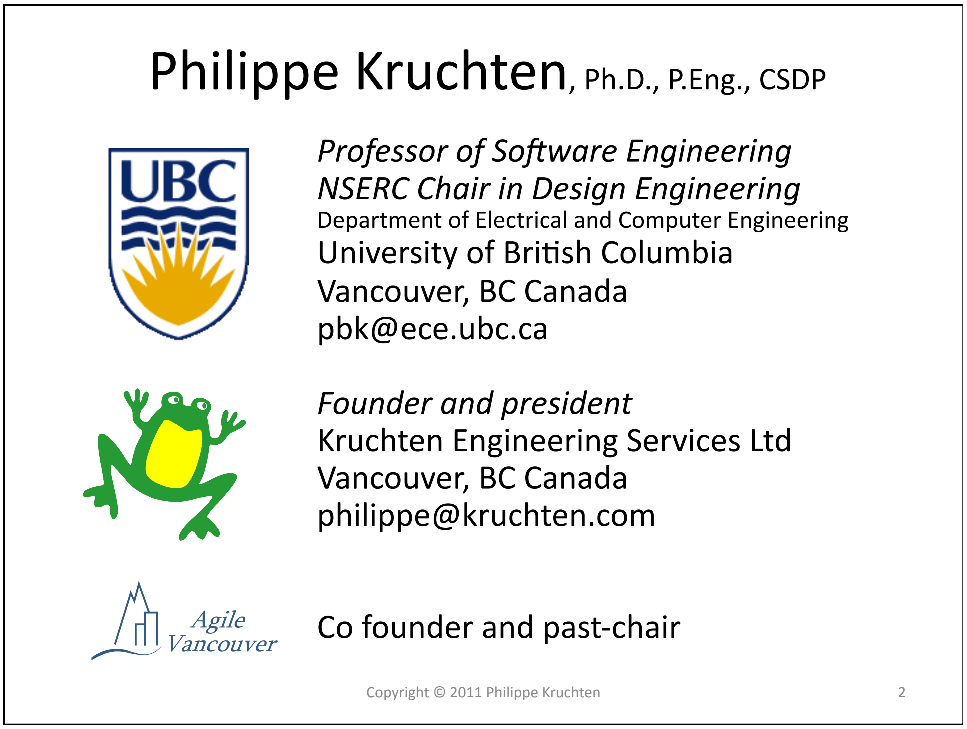


Agility and Architecture

or: What colours is your backlog?

Philippe Kruchten

July 7, 2011



Philippe Kruchten, Ph.D., P.Eng., CSDP

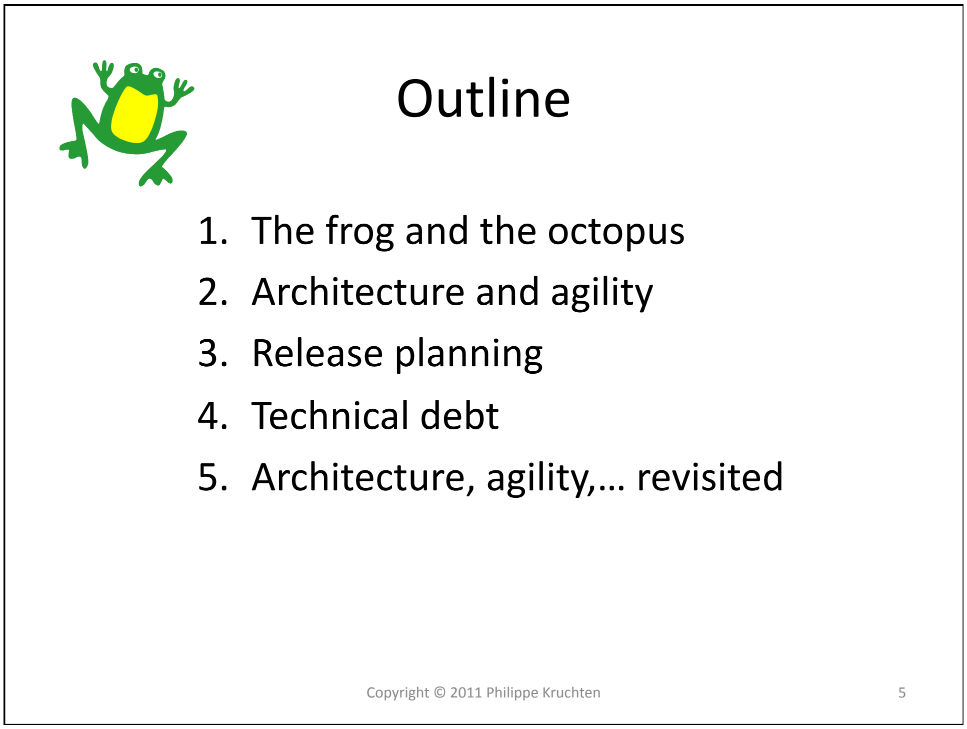
Professor of Software Engineering
NSERC Chair in Design Engineering
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC Canada
pbk@ece.ubc.ca

Founder and president
Kruchten Engineering Services Ltd
Vancouver, BC Canada
philippe@kruchten.com

Agile Vancouver

Co founder and past-chair

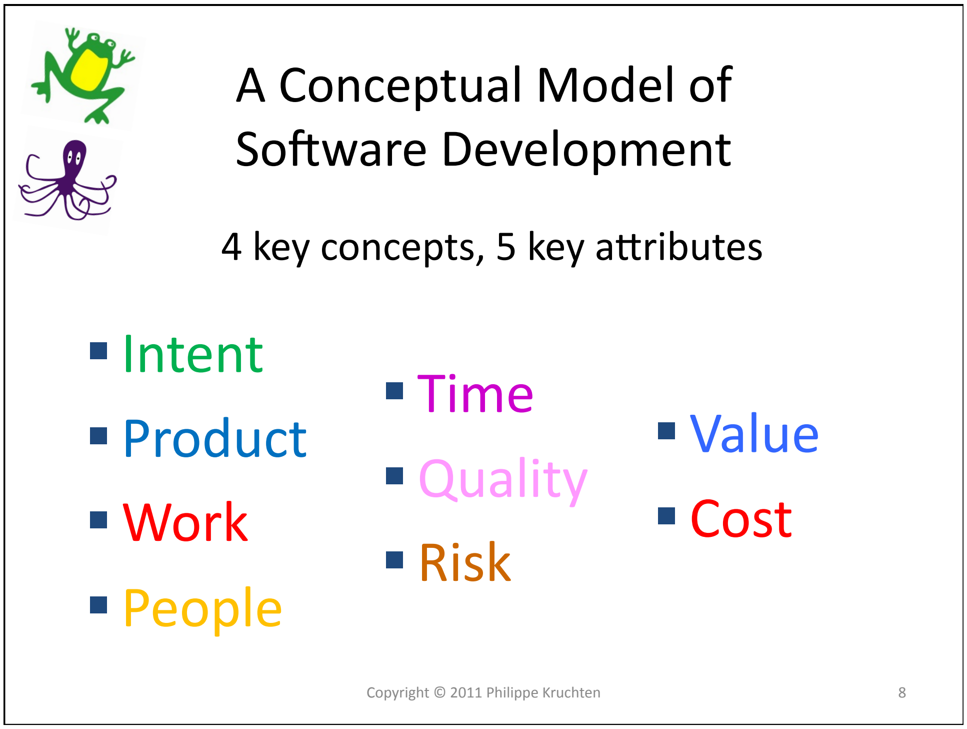
2



Outline

1. The frog and the octopus
2. Architecture and agility
3. Release planning
4. Technical debt
5. Architecture, agility,… revisited

5



A Conceptual Model of Software Development

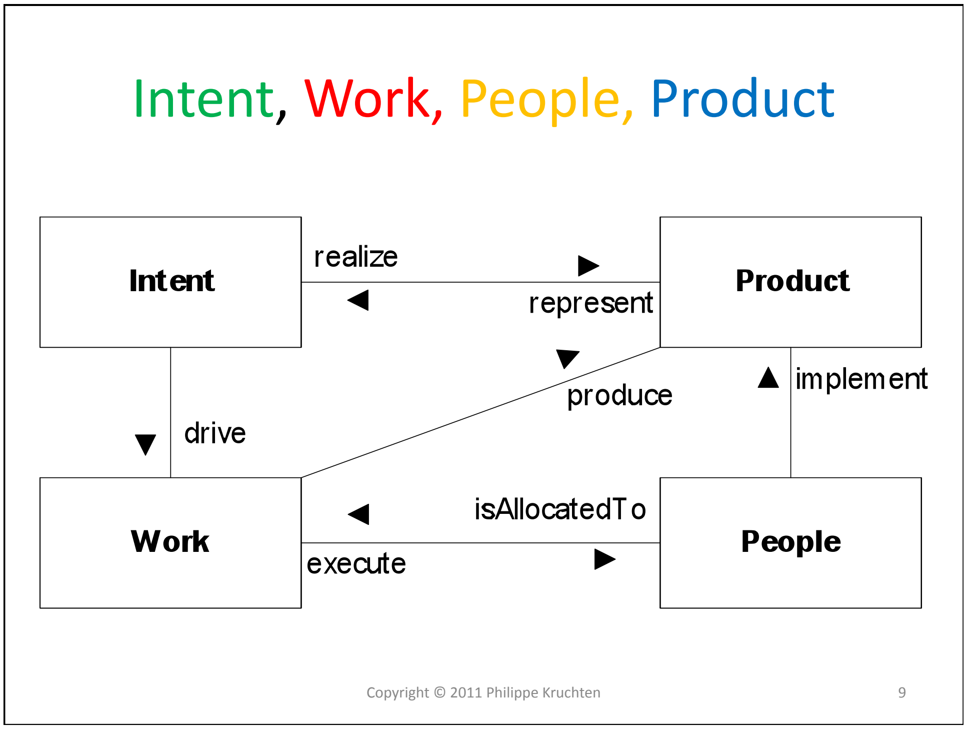
4 key concepts, 5 key attributes

- Intent
- Product
- Work
- People

- Time
- Quality
- Risk

- Value
- Cost

8



Intent, Work, People, Product

Intent

realize ▶

◀ represent

Product

▼ produce

▲ implement

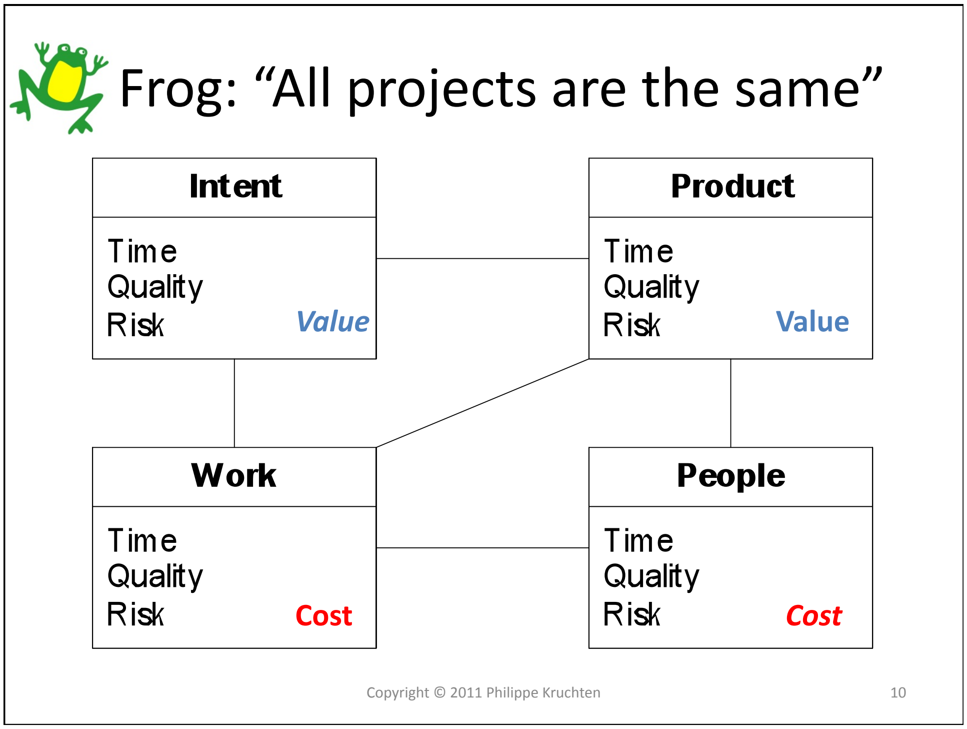
▼ drive

Work

◀ isAllocatedTo

execute ▶

People



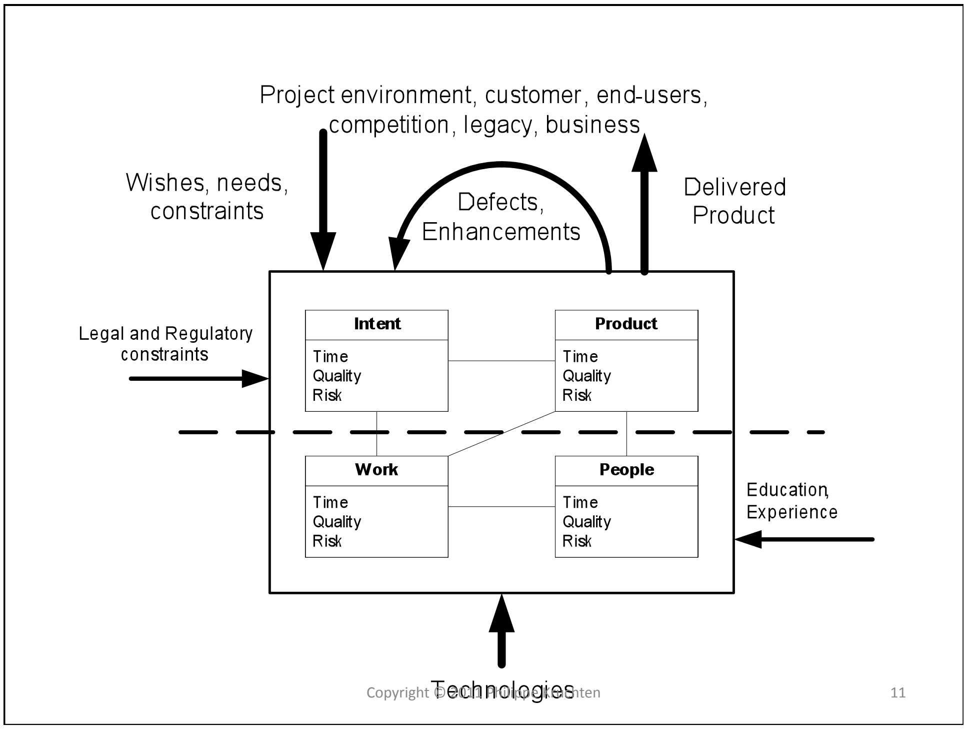
Frog: “All projects are the same”

Intent
Time
Quality
Risk
Value

Product
Time
Quality
Risk
Value

Work
Time
Quality
Risk
Cost

People
Time
Quality
Risk
Cost



Project environment, customer, end-users, competition, legacy, business

Wishes, needs, constraints

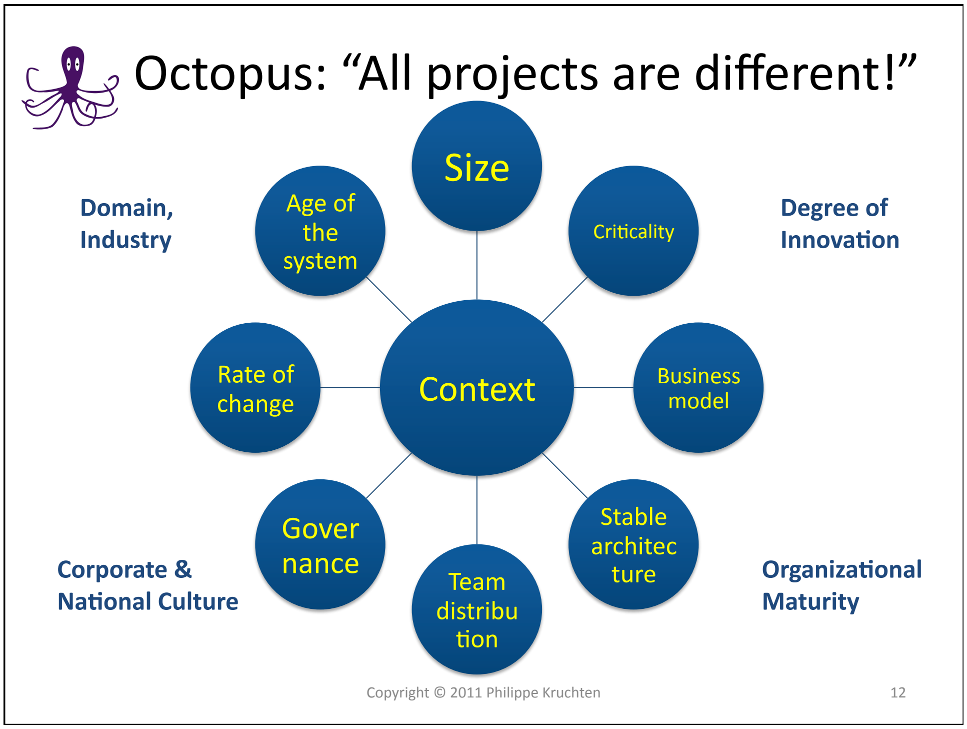
Defects, Enhancements

Delivered Product

Legal and Regulatory constraints

Education, Experience

Technologies



Octopus: “All projects are different!”

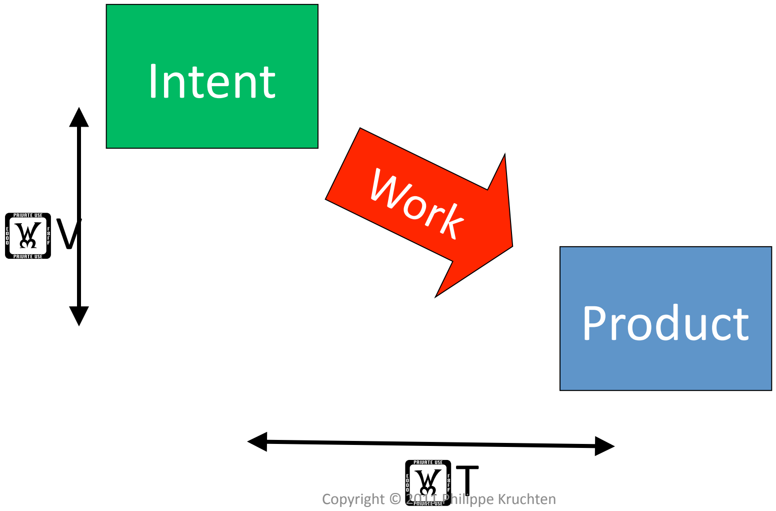
Domain, Industry

Degree of Innovation

Corporate & National Culture

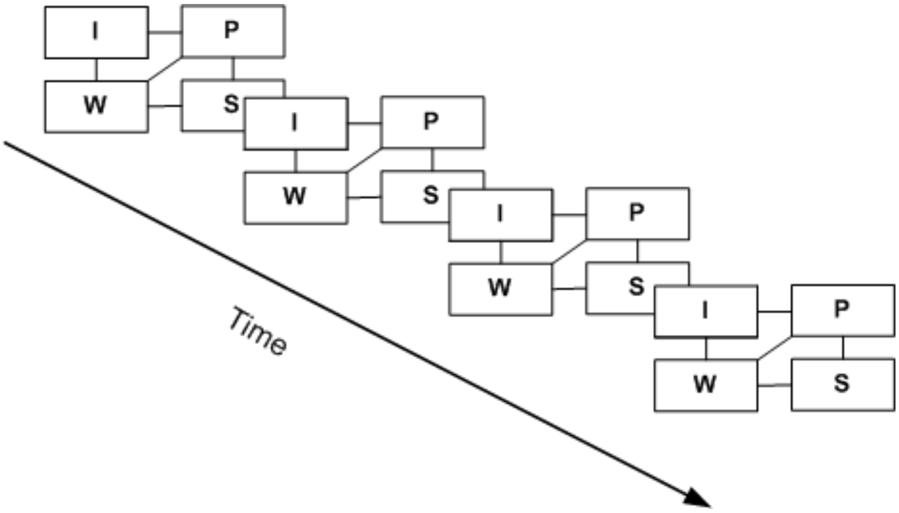
Organizational Maturity

SW Dev. Project: Tension between Intent and Product



Copyright © 2011 Philippe Kruchten 13

Iterations



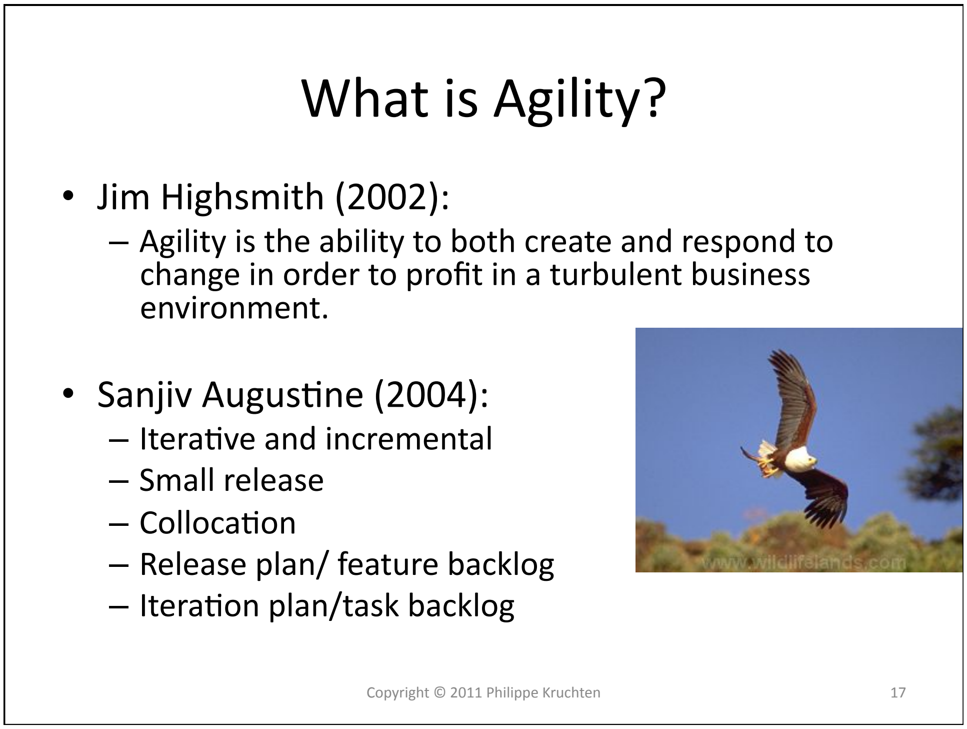
Copyright © 2011 Philippe Kruchten 14

Outline

1. The frog and the octopus
2. **Architecture and agility**
3. Release planning
4. Technical debt
5. Architecture, agility,… revisited

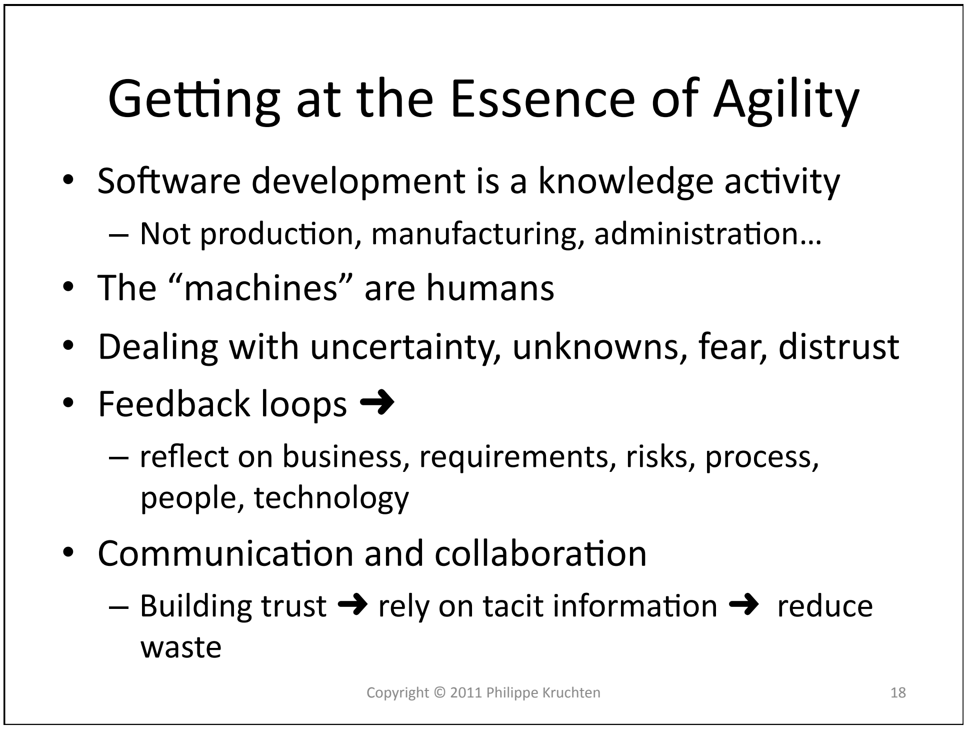
Agile & Architecture? Oil & Water?

- Paradox
- Oxymoron
- Conflict
- Incompatibility



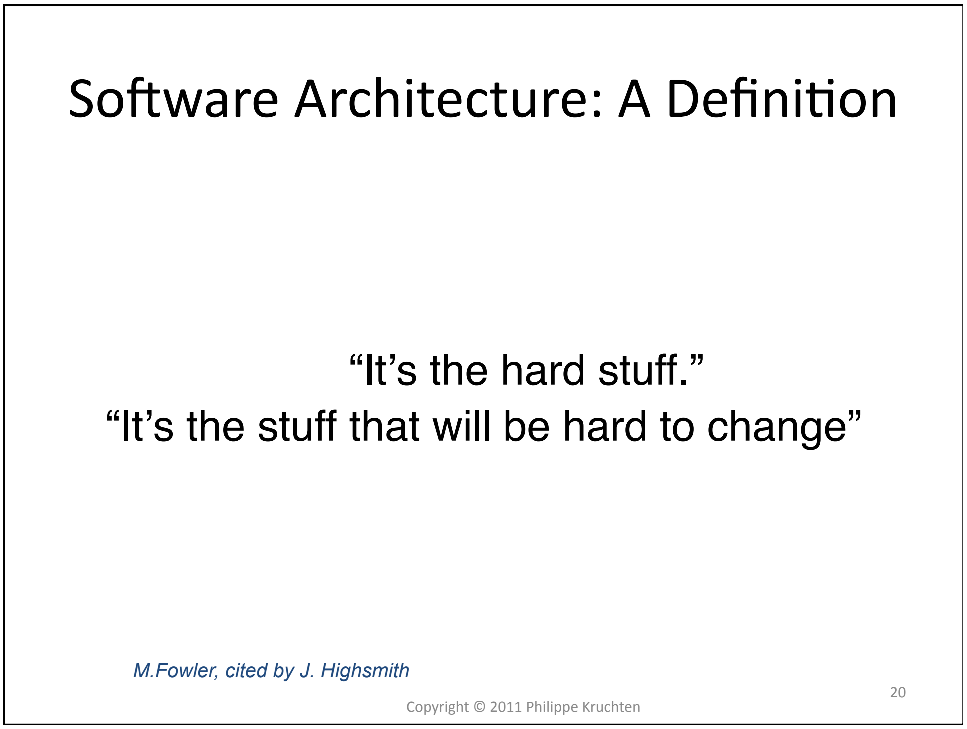
What is Agility?

- Jim Highsmith (2002):
  - Agility is the ability to both create and respond to change in order to profit in a turbulent business environment.
- Sanjiv Augustine (2004):
  - Iterative and incremental
  - Small release
  - Collocation
  - Release plan/ feature backlog
  - Iteration plan/task backlog



Getting at the Essence of Agility

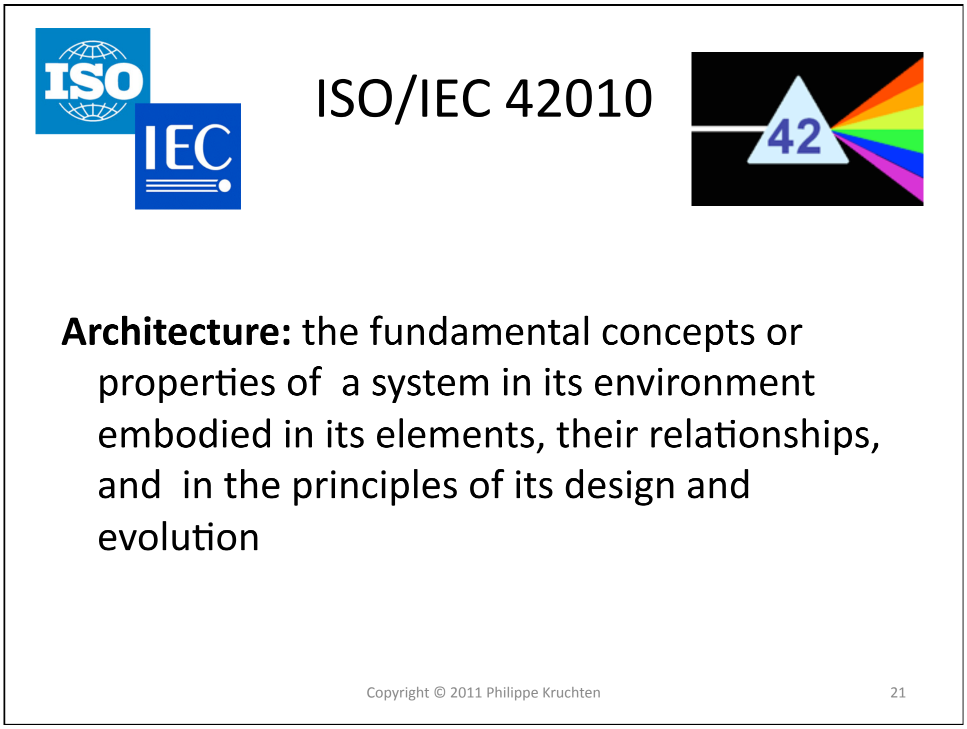
- Software development is a knowledge activity
  - Not production, manufacturing, administration…
- The “machines” are humans
- Dealing with uncertainty, unknowns, fear, distrust
- Feedback loops ➜
  - reflect on business, requirements, risks, process, people, technology
- Communication and collaboration
  - Building trust ➜ rely on tacit information ➜ reduce waste



Software Architecture: A Definition

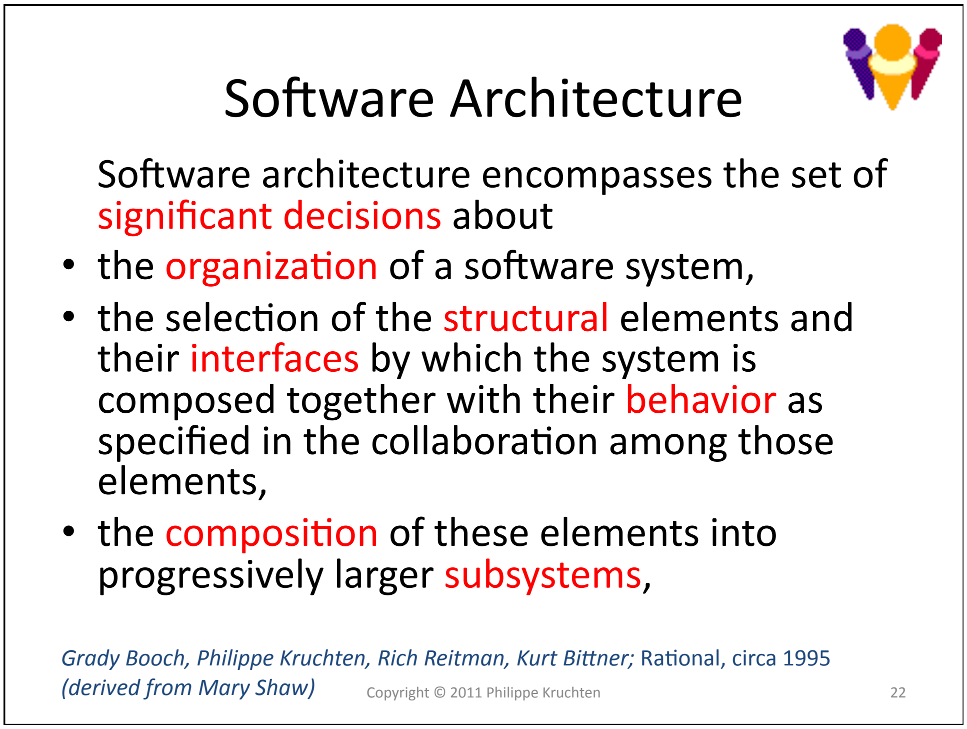
“It’s the hard stuff.”
“It’s the stuff that will be hard to change”

M.Fowler, cited by J. Highsmith



ISO/IEC 42010

Architecture: the fundamental concepts or properties of a system in its environment embodied in its elements, their relationships, and in the principles of its design and evolution

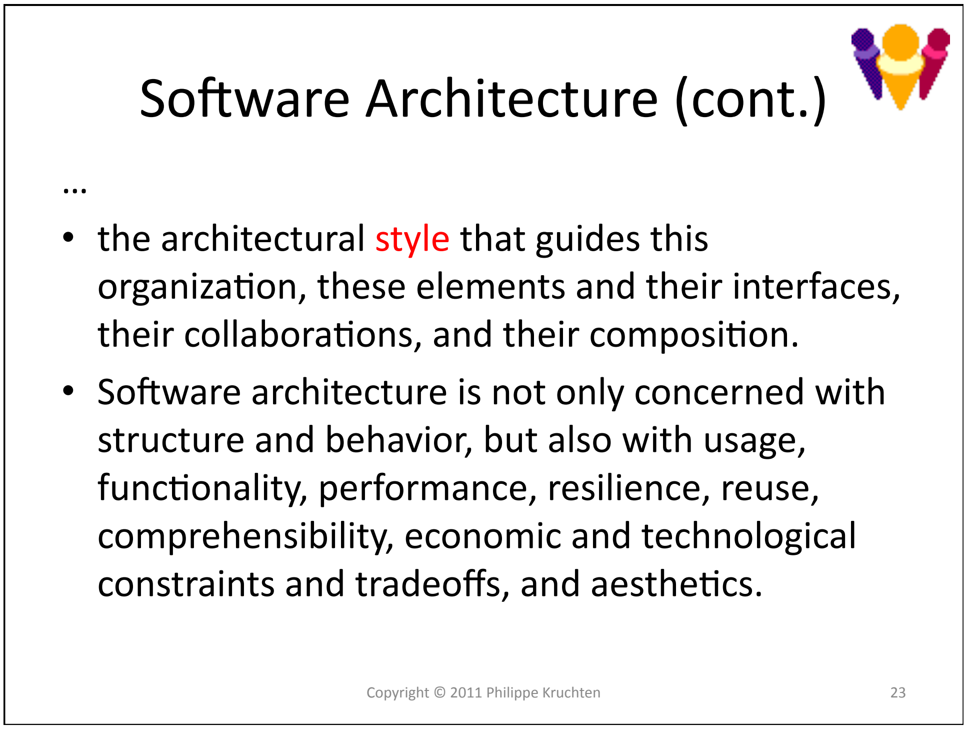


Software Architecture

Software architecture encompasses the set of significant decisions about

- the organization of a software system,
- the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements,
- the composition of these elements into progressively larger subsystems,

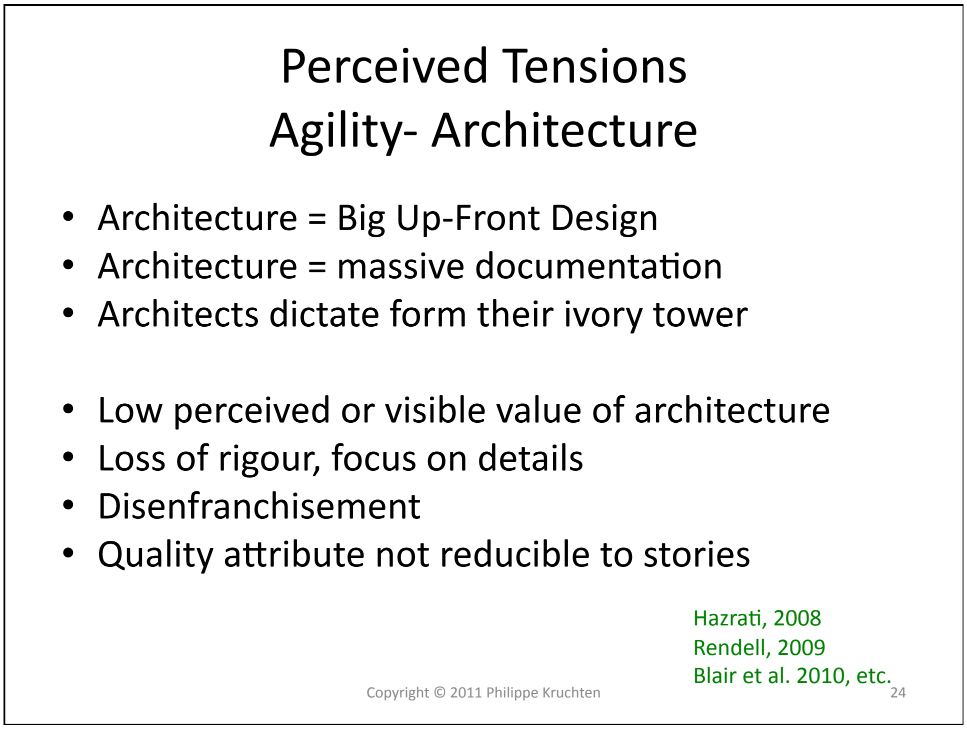
Grady Booch, Philippe Kruchten, Rich Reitman, Kurt Bittner; Rational, circa 1995
(derived from Mary Shaw)



Software Architecture (cont.)

…

- the architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition.
- Software architecture is not only concerned with structure and behavior, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints and tradeoffs, and aesthetics.

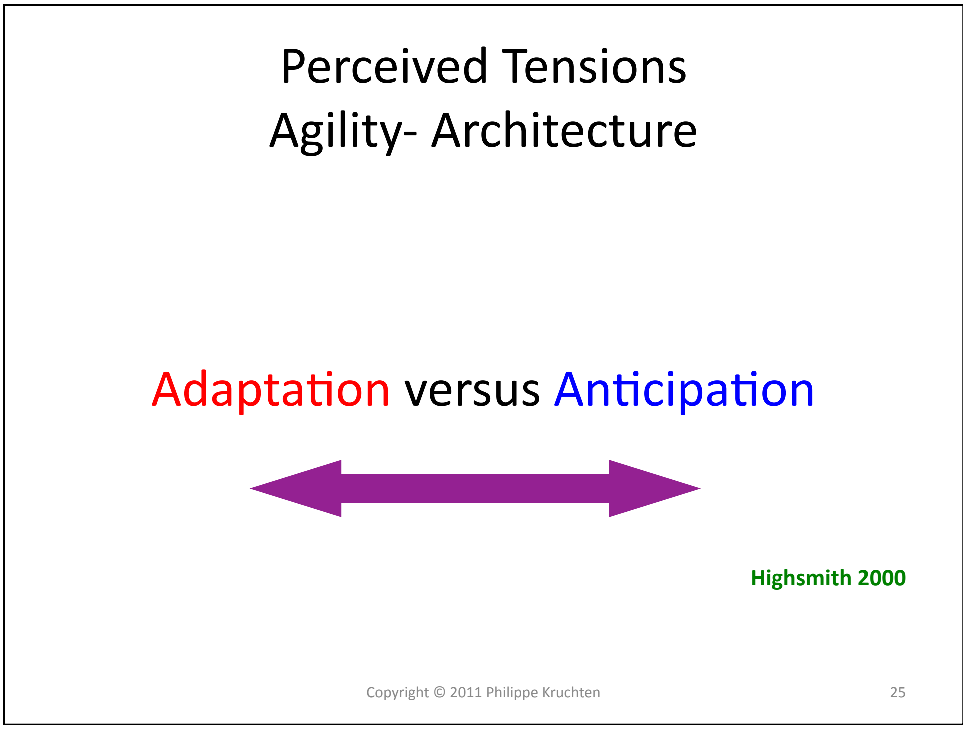


Perceived Tensions Agility- Architecture

- Architecture = Big Up-Front Design
- Architecture = massive documentation
- Architects dictate form their ivory tower

- Low perceived or visible value of architecture
- Loss of rigour, focus on details
- Disenfranchisement
- Quality attribute not reducible to stories

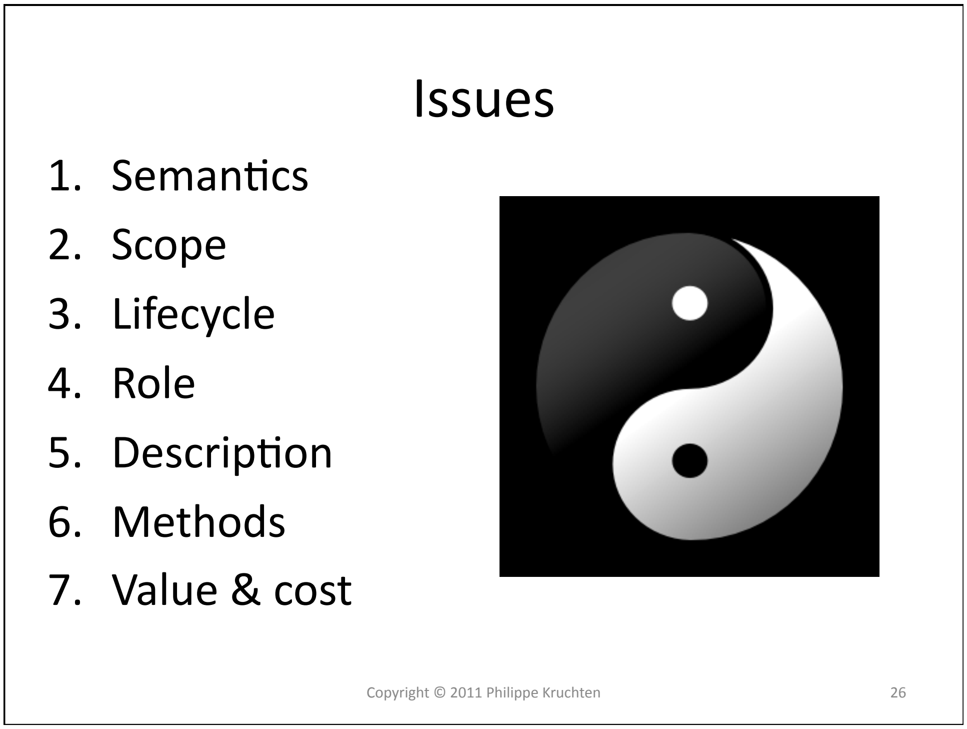
Hazrati, 2008
Rendell, 2009
Blair et al. 2010, etc.



Perceived Tensions Agility- Architecture

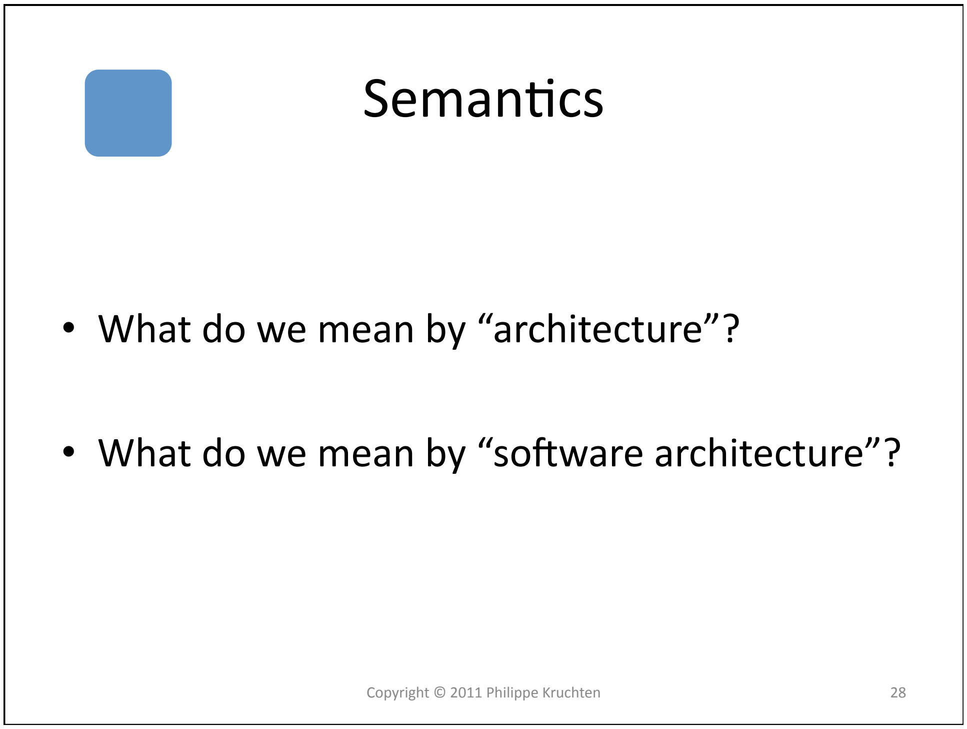
Adaptation versus Anticipation

Highsmith 2000



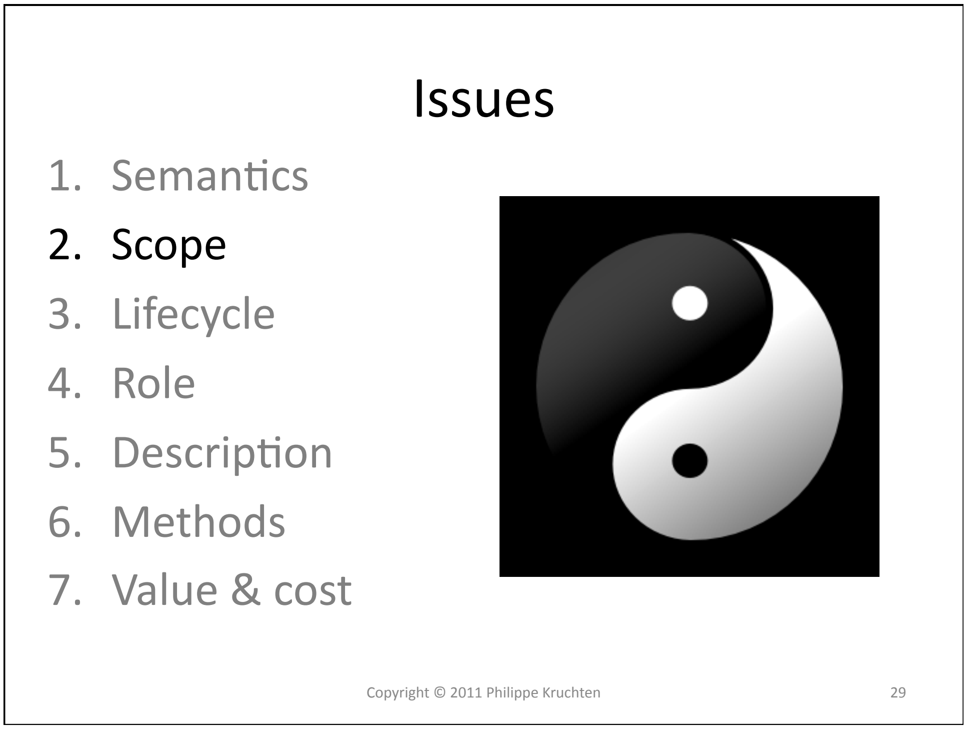
Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost



Semantics

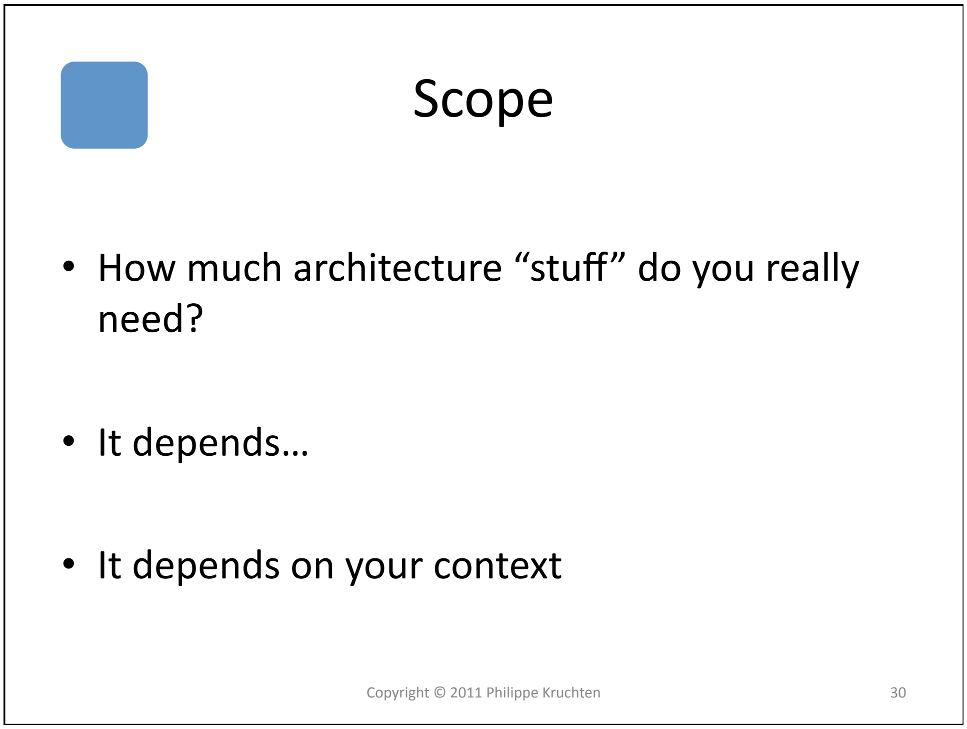
- What do we mean by “architecture”?
- What do we mean by “software architecture”?



Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

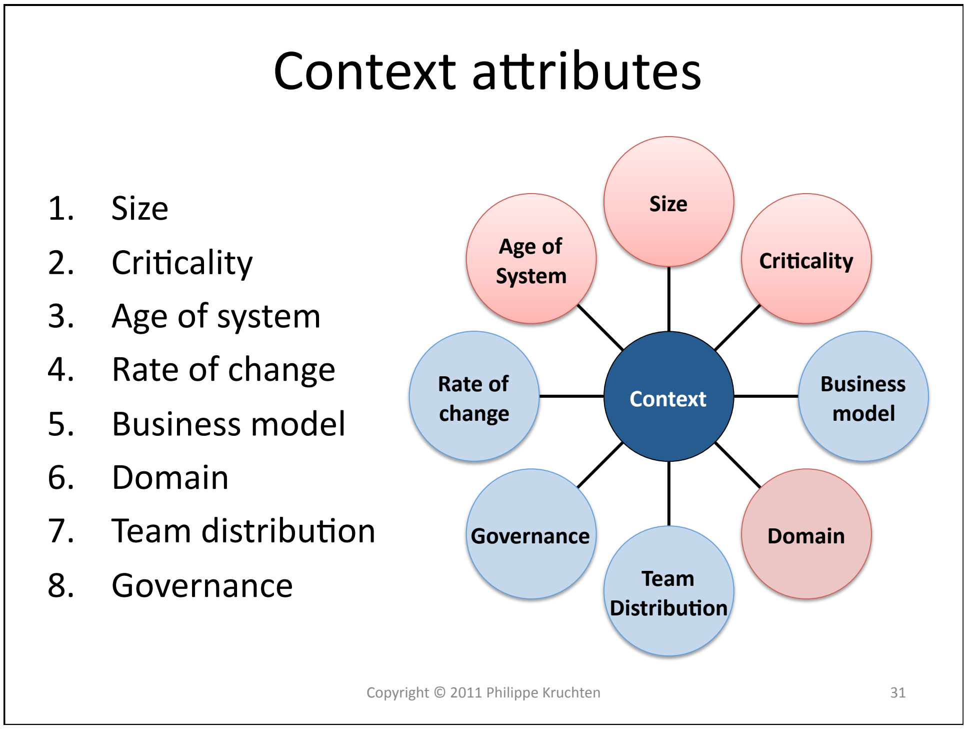
29



Scope

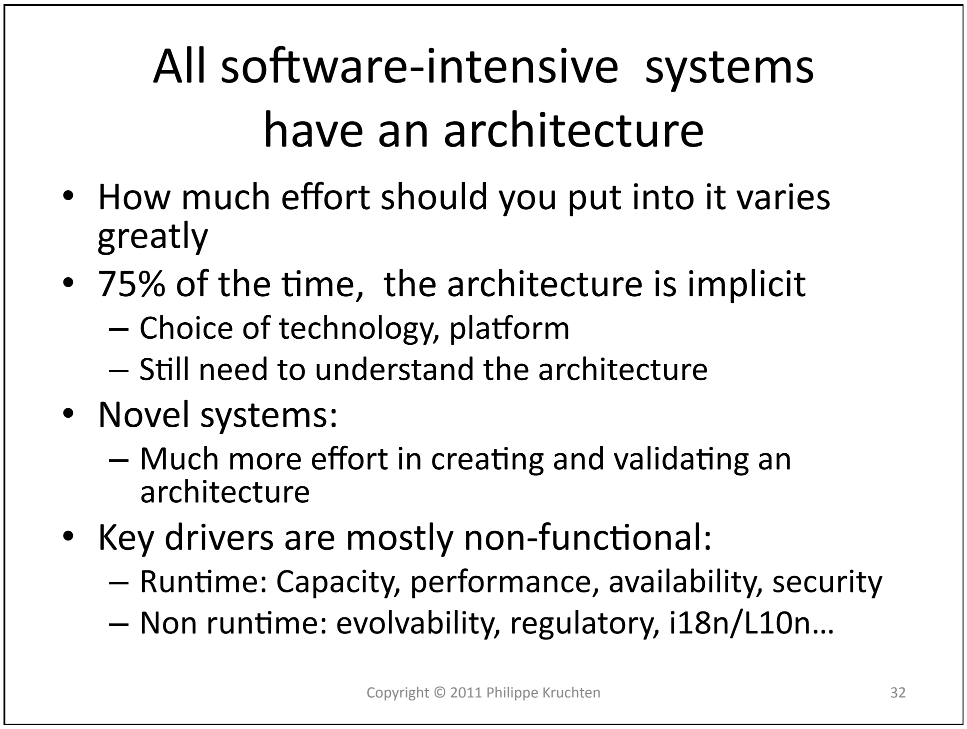
- How much architecture “stuff” do you really need?
- It depends…
- It depends on your context

30



Context attributes

1. Size
2. Criticality
3. Age of system
4. Rate of change
5. Business model
6. Domain
7. Team distribution
8. Governance



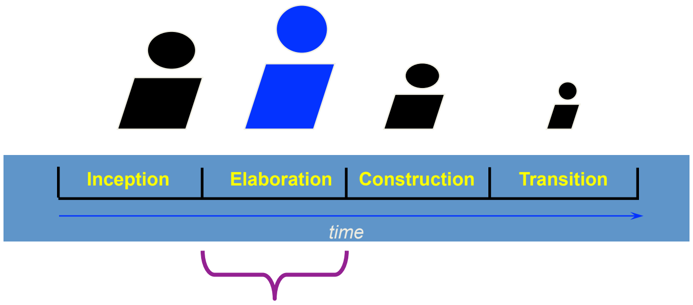
All software-intensive systems have an architecture

- How much effort should you put into it varies greatly
- 75% of the time, the architecture is implicit
  - Choice of technology, platform
  - Still need to understand the architecture
- Novel systems:
  - Much more effort in creating and validating an architecture
- Key drivers are mostly non-functional:
  - Runtime: Capacity, performance, availability, security
  - Non runtime: evolvability, regulatory, i18n/L10n…

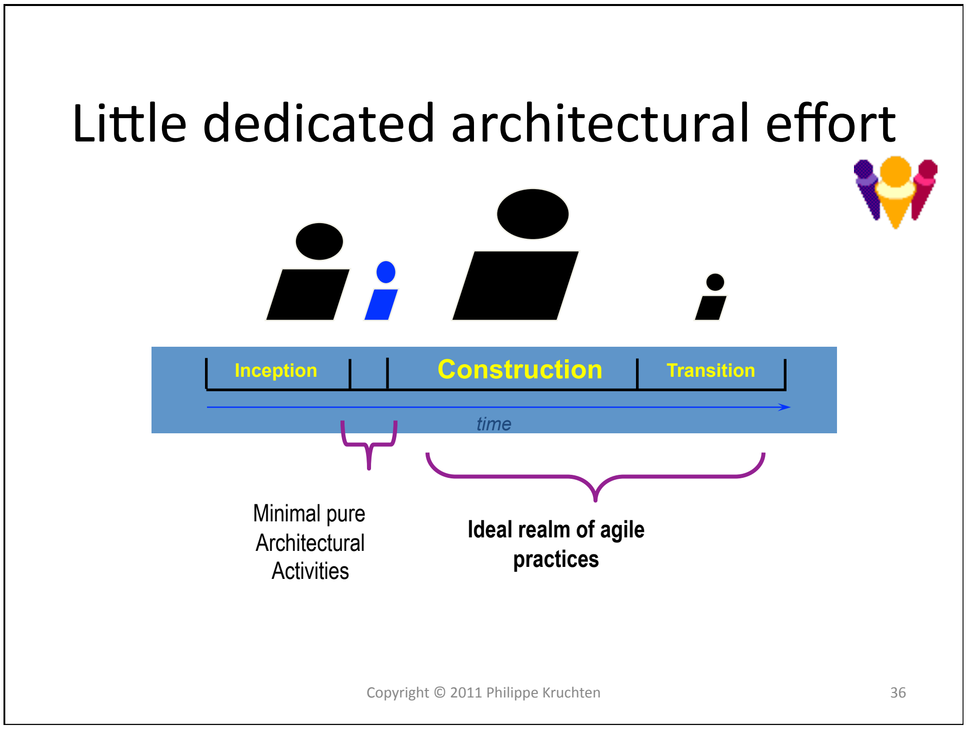
Lifecycle

- When does architectural activities take place?
- The evil of “BUFD” = Big Up-Front Design
- “Defer decisions to the last responsible moment”
- YAGNI = You Ain’t Gonna Need It
- Refactor!

Architectural Effort During the Lifecycle



Majority of architectural design activities

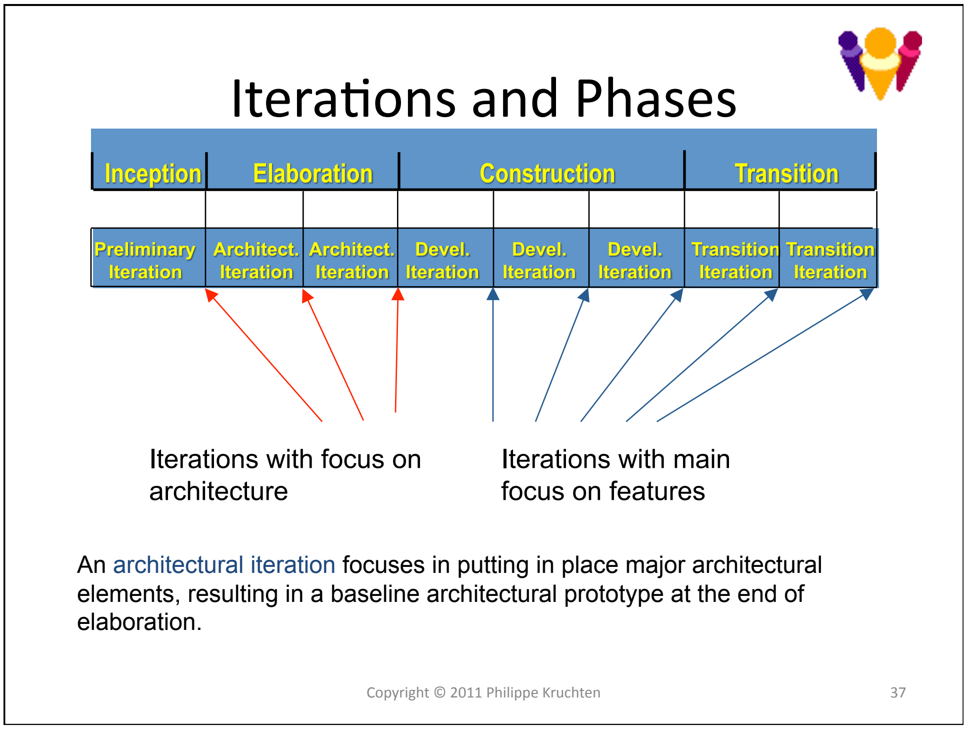


Little dedicated architectural effort

Minimal pure Architectural Activities

Ideal realm of agile practices

36



Iterations and Phases

Inception	Elaboration		Construction			Transition	
Preliminary Iteration	Architect. Iteration	Architect. Iteration	Devel. Iteration	Devel. Iteration	Devel. Iteration	Transition Iteration	Transition Iteration

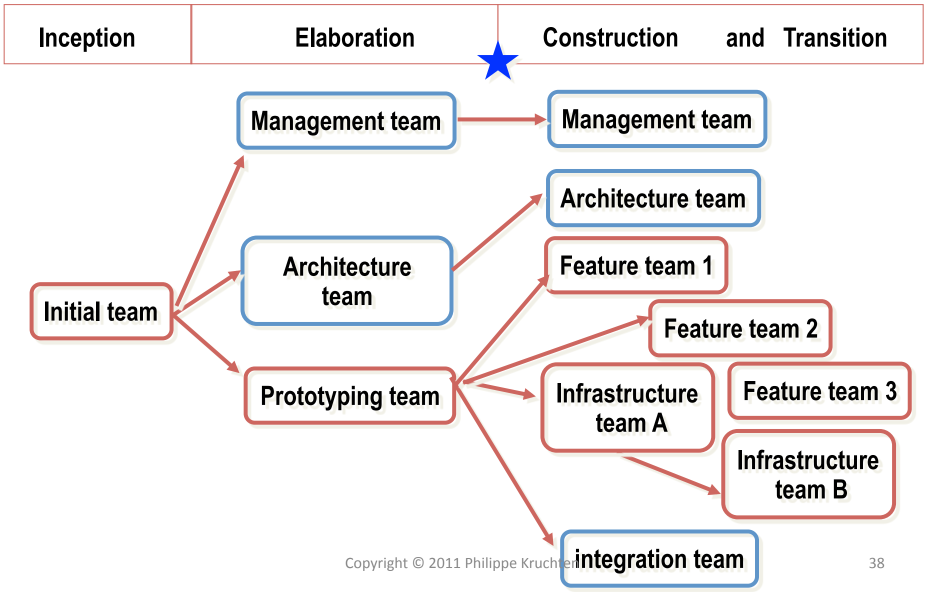
Iterations with focus on architecture

Iterations with main focus on features

An architectural iteration focuses in putting in place major architectural elements, resulting in a baseline architectural prototype at the end of elaboration.

37

Team Structure over Time (Very Large)



Inception

Elaboration

Construction and Transition

Management team

Management team

Architecture team

Architecture team

Initial team

Feature team 1

Feature team 2

Prototyping team

Infrastructure team A

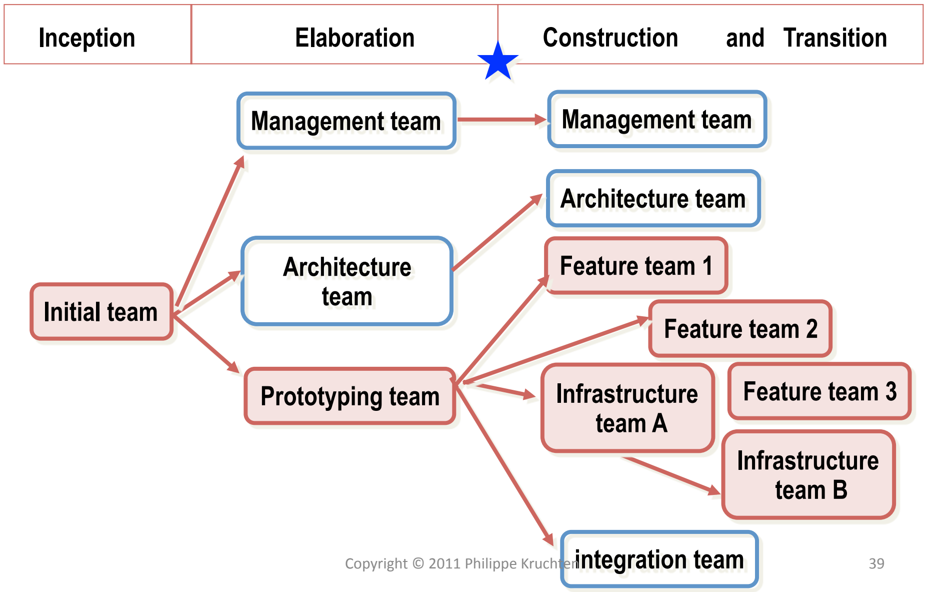
Feature team 3

Infrastructure team B

integration team

38

Teams using agile development practices



Inception

Elaboration

Construction and Transition

Management team

Management team

Architecture team

Architecture team

Initial team

Feature team 1

Feature team 2

Prototyping team

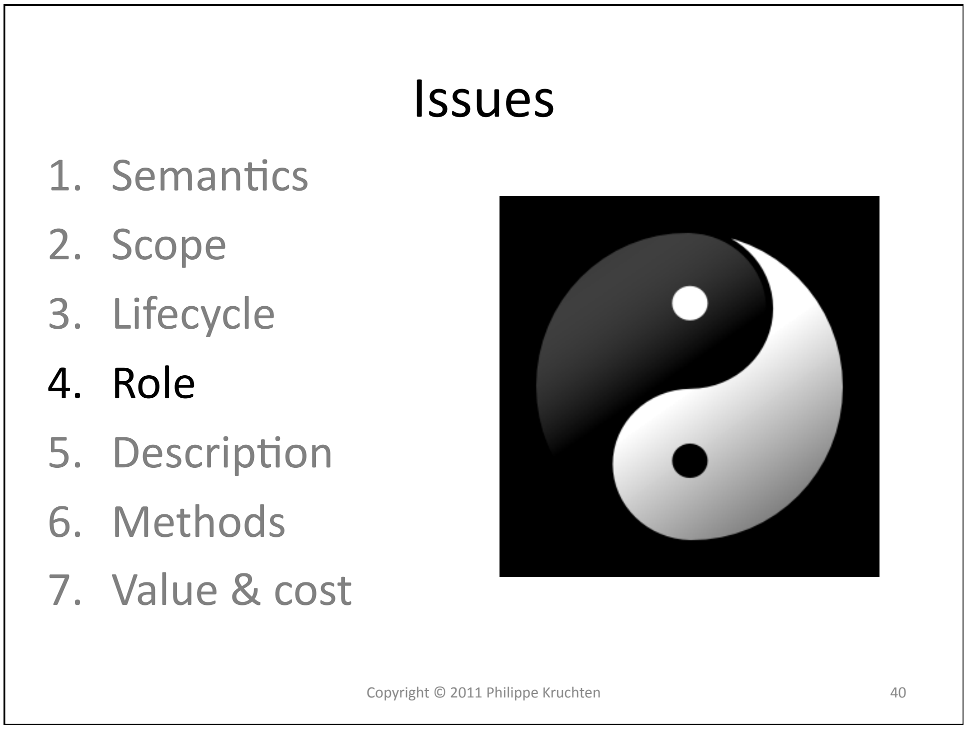
Infrastructure team A

Feature team 3

Infrastructure team B

integration team

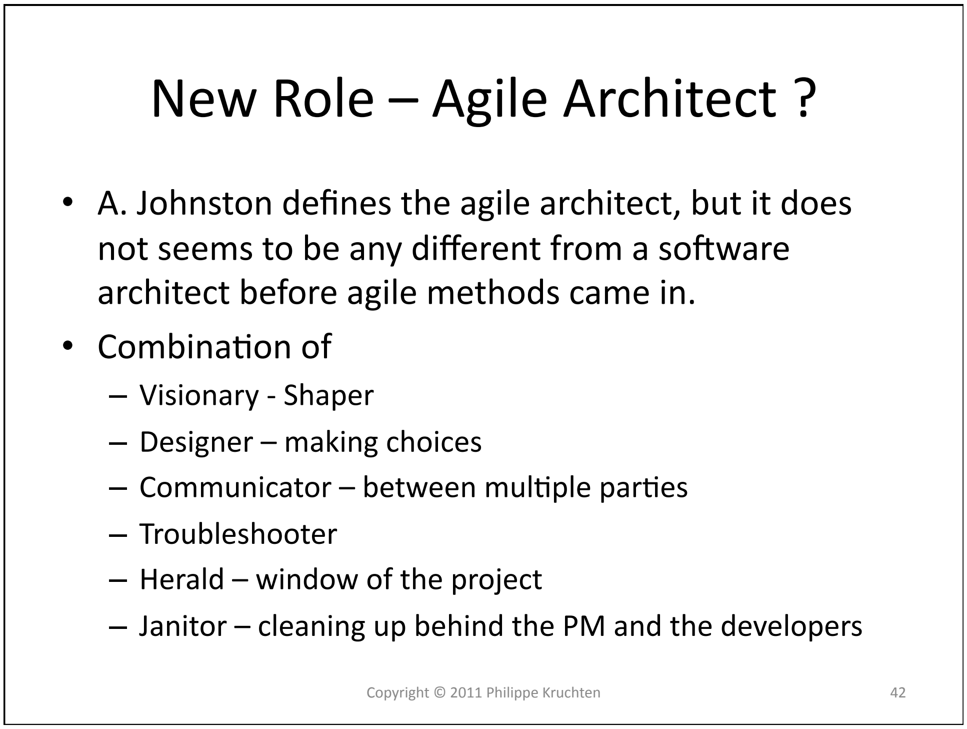
39



Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

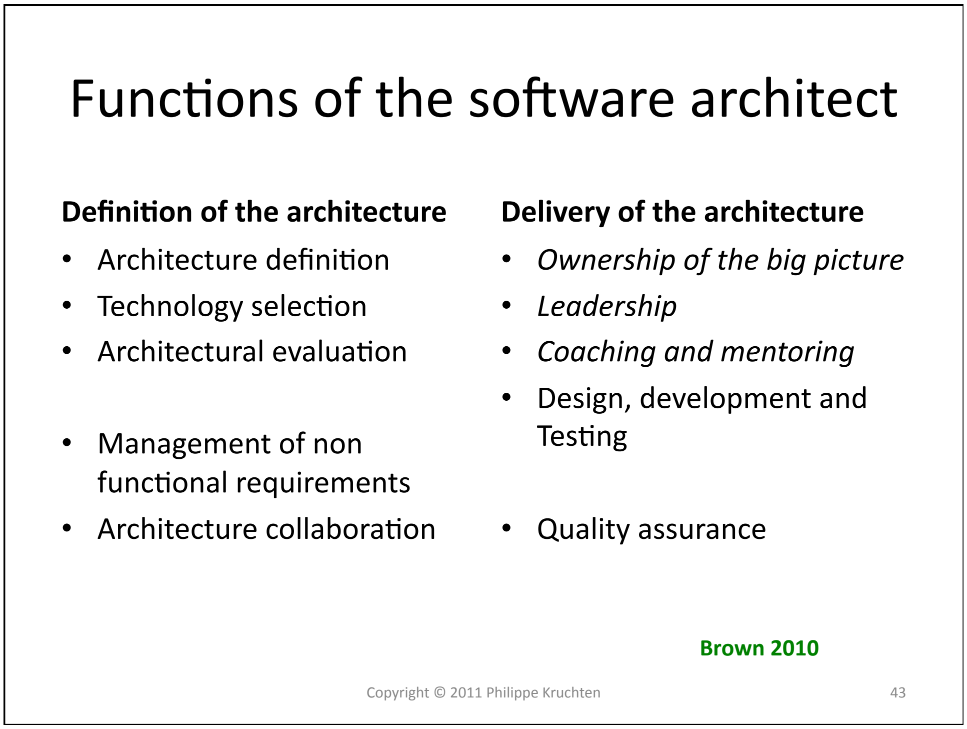
40



New Role – Agile Architect ?

- A. Johnston defines the agile architect, but it does not seems to be any different from a software architect before agile methods came in.
- Combination of
  - Visionary - Shaper
  - Designer – making choices
  - Communicator – between multiple parties
  - Troubleshooter
  - Herald – window of the project
  - Janitor – cleaning up behind the PM and the developers

42



Functions of the software architect

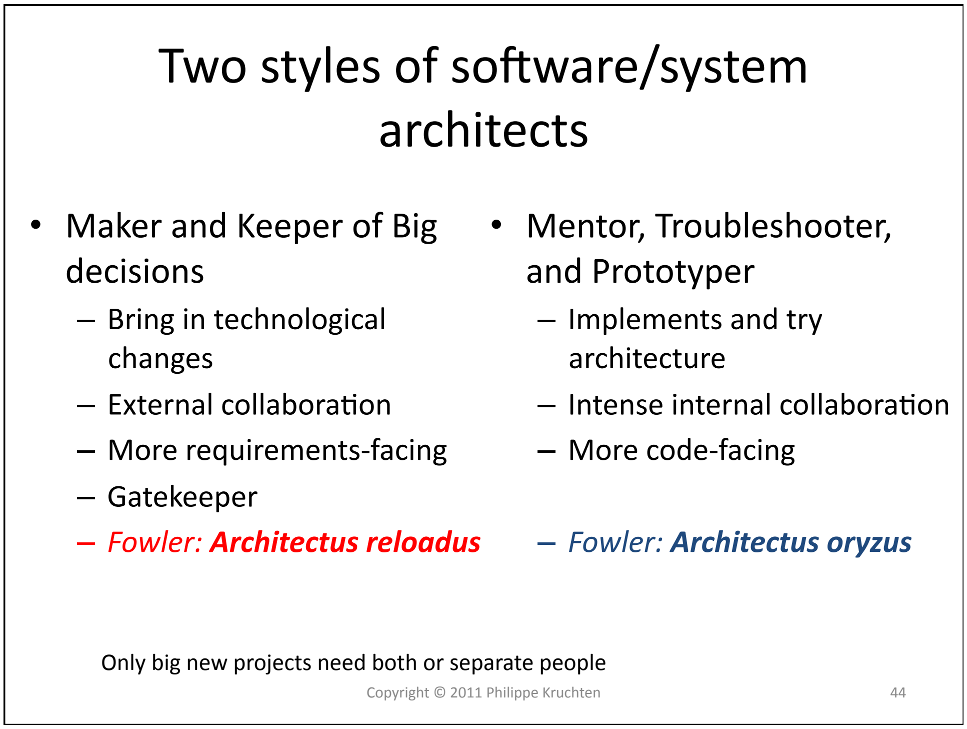
Definition of the architecture

- Architecture definition
- Technology selection
- Architectural evaluation
- Management of non functional requirements
- Architecture collaboration

Delivery of the architecture

- *Ownership of the big picture*
- *Leadership*
- *Coaching and mentoring*
- Design, development and Testing
- Quality assurance

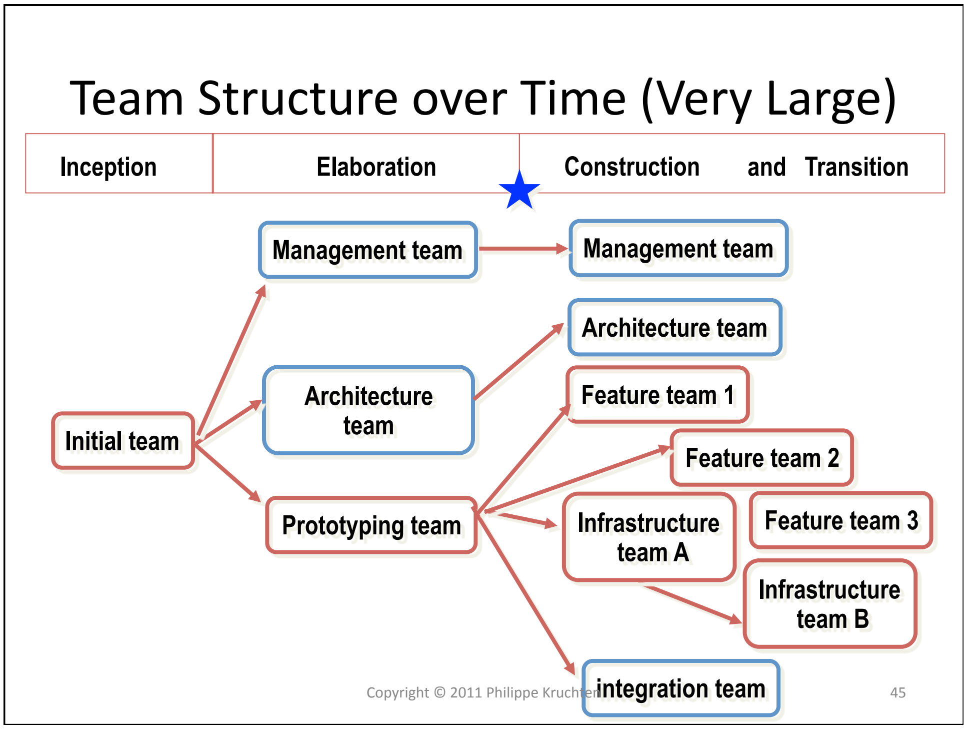
Brown 2010



Two styles of software/system architects

- Maker and Keeper of Big decisions
  - Bring in technological changes
  - External collaboration
  - More requirements-facing
  - Gatekeeper
  - *Fowler:* ***Architectus reloadus***
- Mentor, Troubleshooter, and Prototyper
  - Implements and try architecture
  - Intense internal collaboration
  - More code-facing
  - *Fowler:* ***Architectus oryzus***

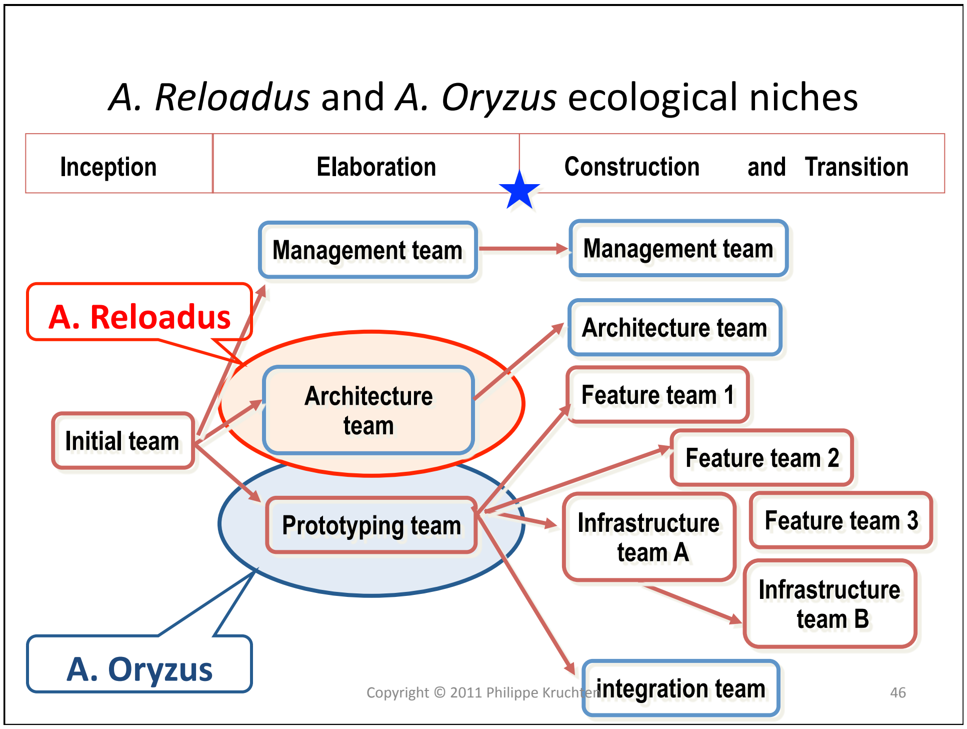
Only big new projects need both or separate people



Team Structure over Time (Very Large)

Inception | Elaboration | Construction and Transition

45

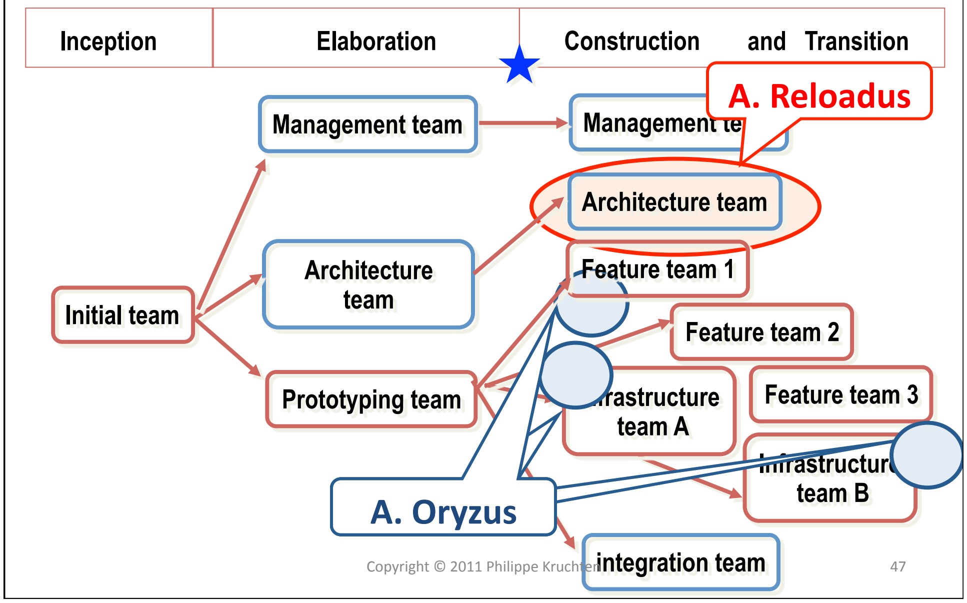


A. Reloadus and *A. Oryzus* ecological niches

Inception | Elaboration | Construction and Transition

46

A. Reloadus and *A. Oryzus* ecological niches



Inception

Elaboration

Construction and Transition

A. Reloadus

Management team

Management te

Architecture team

Initial team

Architecture team

Feature team 1

Feature team 2

Prototyping team

rastructure team A

Feature team 3

Infrastructur team B

A. Oryzus

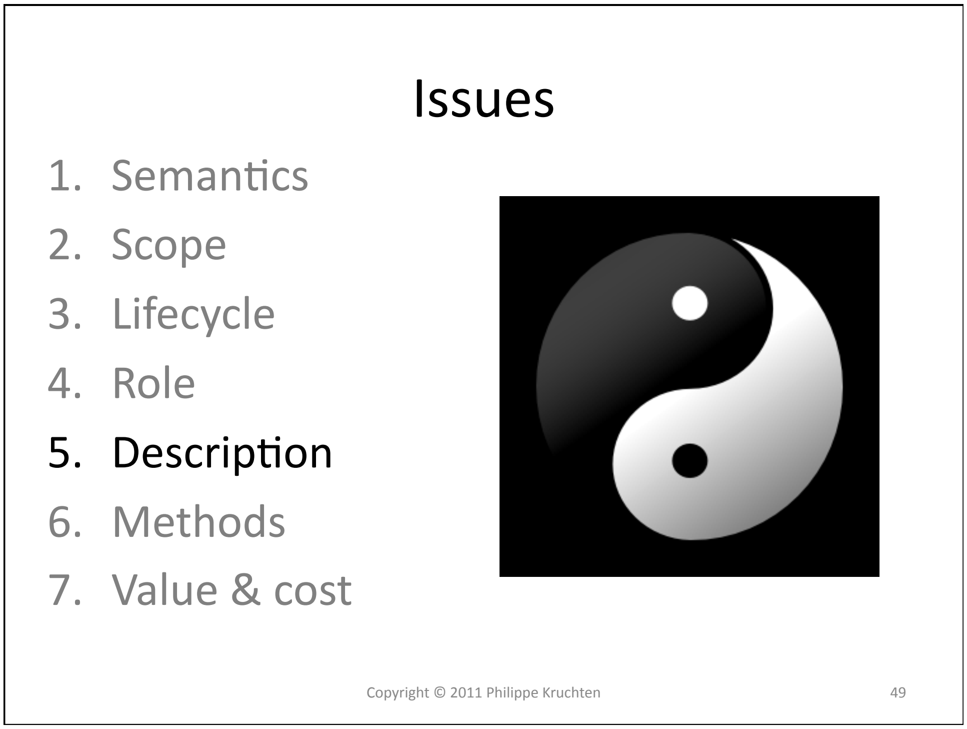
integration team

47

Role

Architecture owner

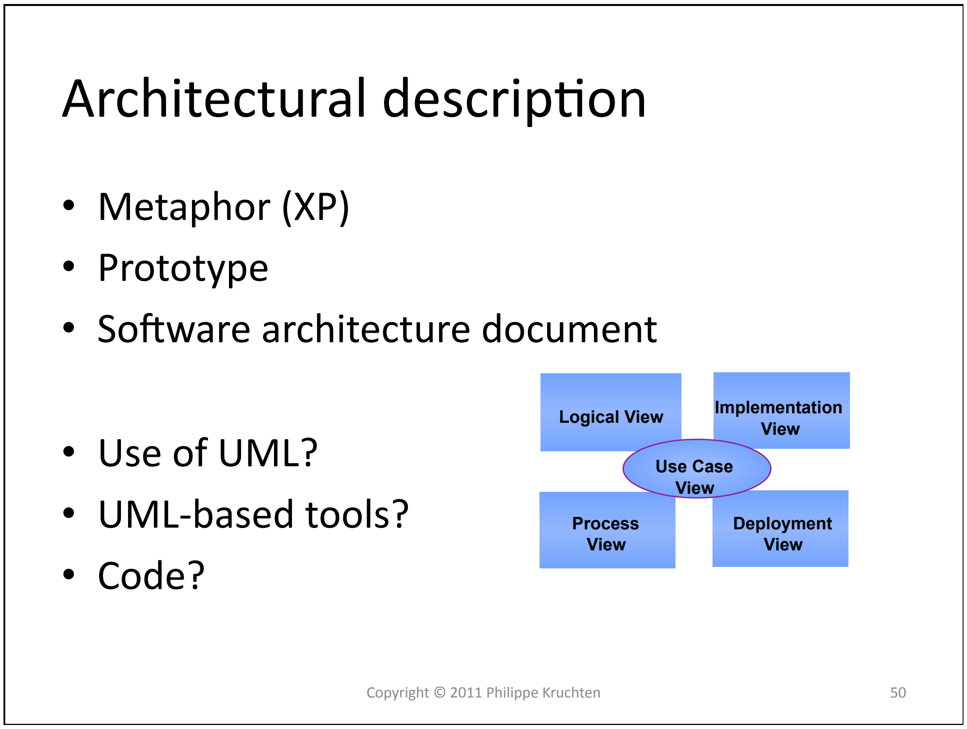
48



Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

49

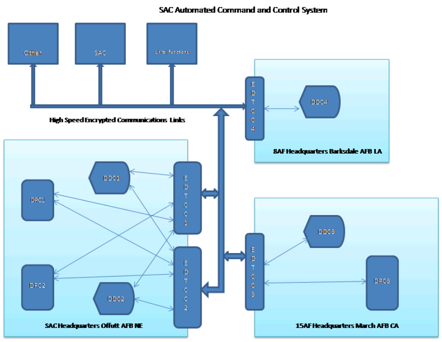


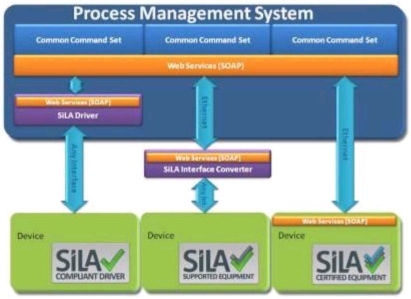
Architectural description

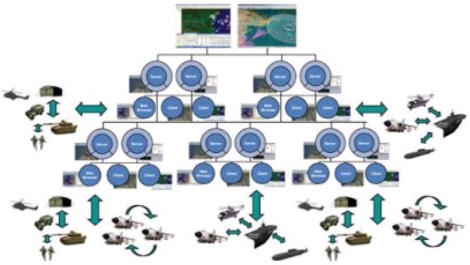
- Metaphor (XP)
- Prototype
- Software architecture document

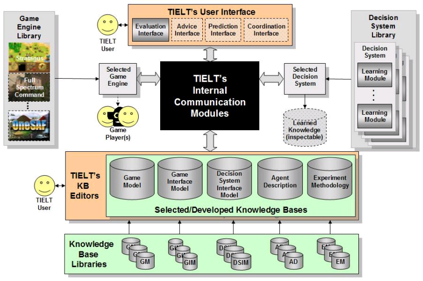
- Use of UML?
- UML-based tools?
- Code?

50


SAC Automated Command and Control System
Other
SAC
Units/Functions
High Speed Encrypted Communications Links
EDTCC4
DDC4
8AF Headquarters Barksdale AFB LA
DDC1
DPC1
EDTCC1
EDTCC2
DPC2
DDC2
SAC Headquarters Offutt AFB NE
EDTCC3
DDC3
DPC3
15AF Headquarters March AFB CA





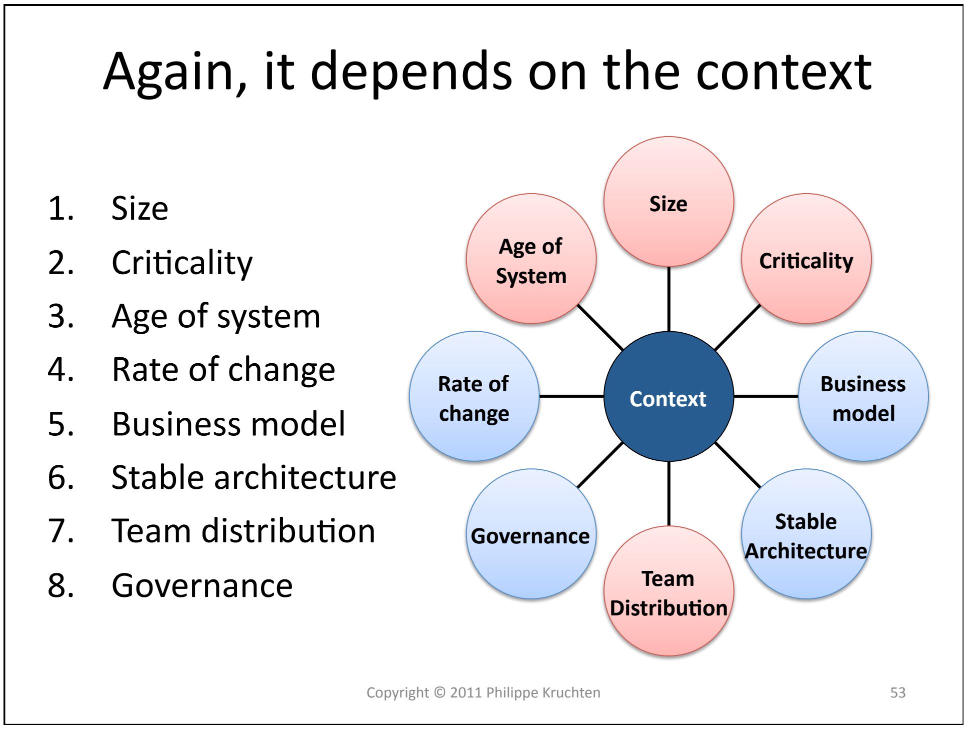


UML 2.0

- A notation
- Better “box and arrows”
- Crisper semantics
- Almost an ADL ?

UNIFIED
MODELING
LANGUAGE

- Model-driven design,
- Model-driven architecture.



Again, it depends on the context

1. Size
2. Criticality
3. Age of system
4. Rate of change
5. Business model
6. Stable architecture
7. Team distribution
8. Governance

Context

Size

Criticality

Business model

Stable Architecture

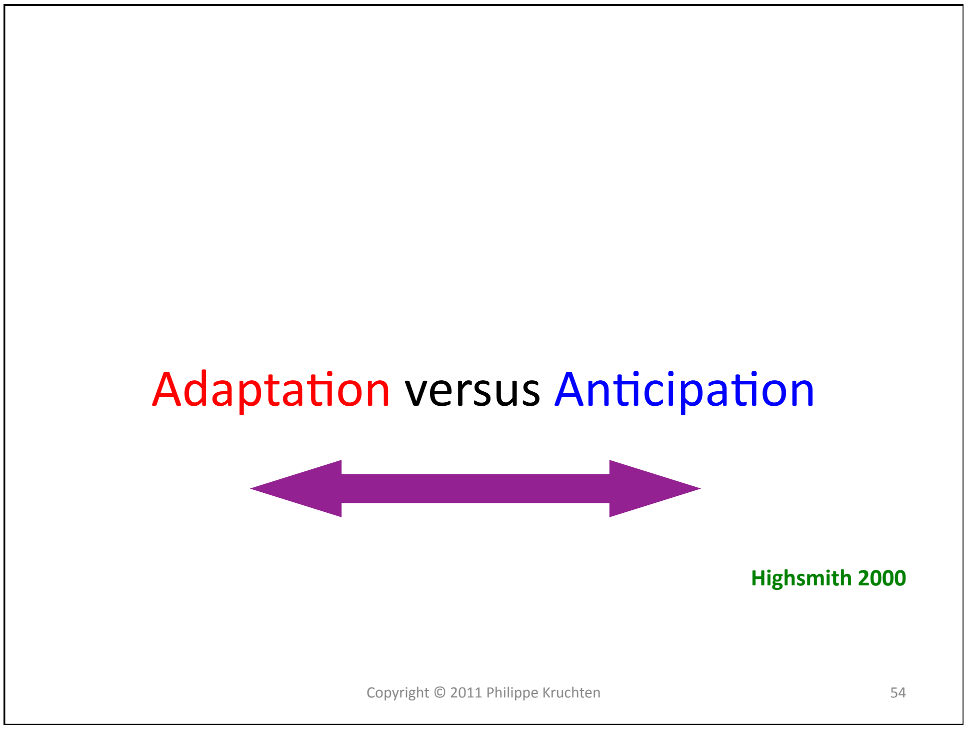
Team Distribution

Governance

Rate of change

Age of System

53



Adaptation versus Anticipation

Highsmith 2000

54

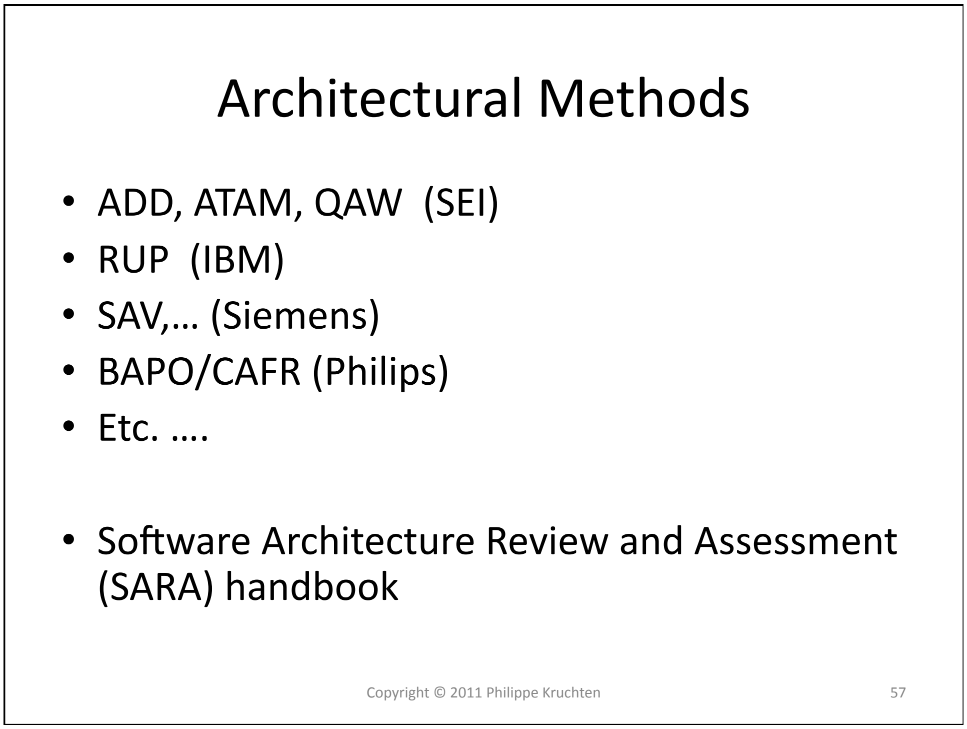
Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

Architectural design methods

- Many agile developers do not know (much) about architectural design
- Agile methods have no explicit guidance for architecture
  - Metaphor in XP
  - Technical activities in scrum
- Relate this to Semantics and Scope issue

- May have to get above the code level

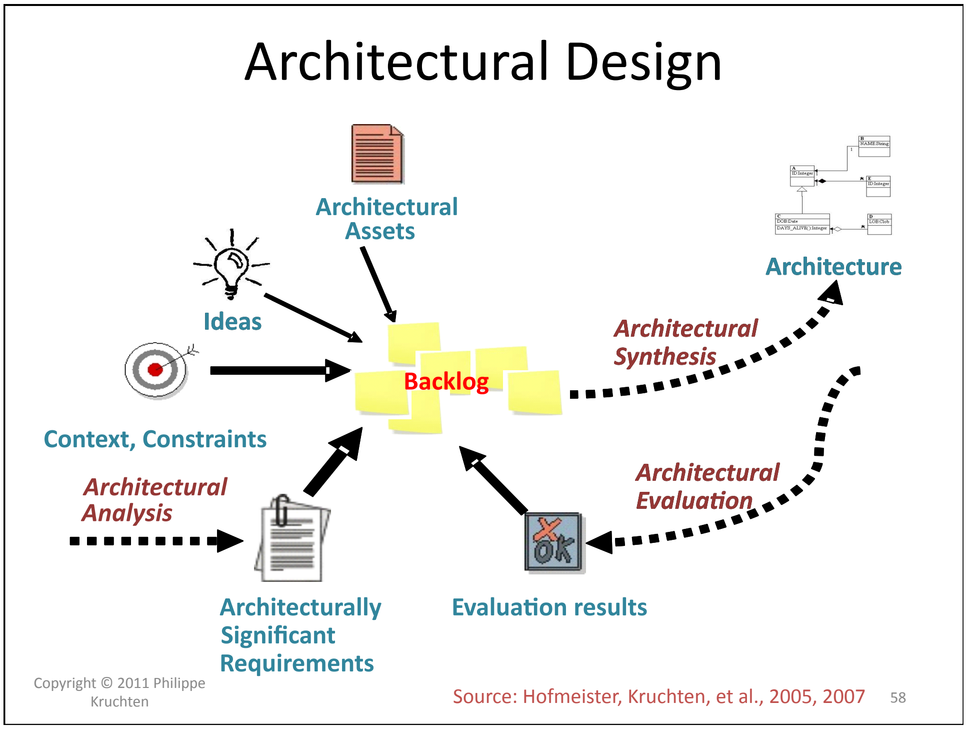


Architectural Methods

- ADD, ATAM, QAW (SEI)
- RUP (IBM)
- SAV,… (Siemens)
- BAPO/CAFR (Philips)
- Etc. ….

- Software Architecture Review and Assessment (SARA) handbook

57



Architectural Design

Architectural Assets

Architecture

Ideas

Architectural Synthesis

Backlog

Context, Constraints

Architectural Analysis

Architectural Evaluation

Architecturally Significant Requirements

Evaluation results

Source: Hofmeister, Kruchten, et al., 2005, 2007 58

Iterative Architecture Refinement

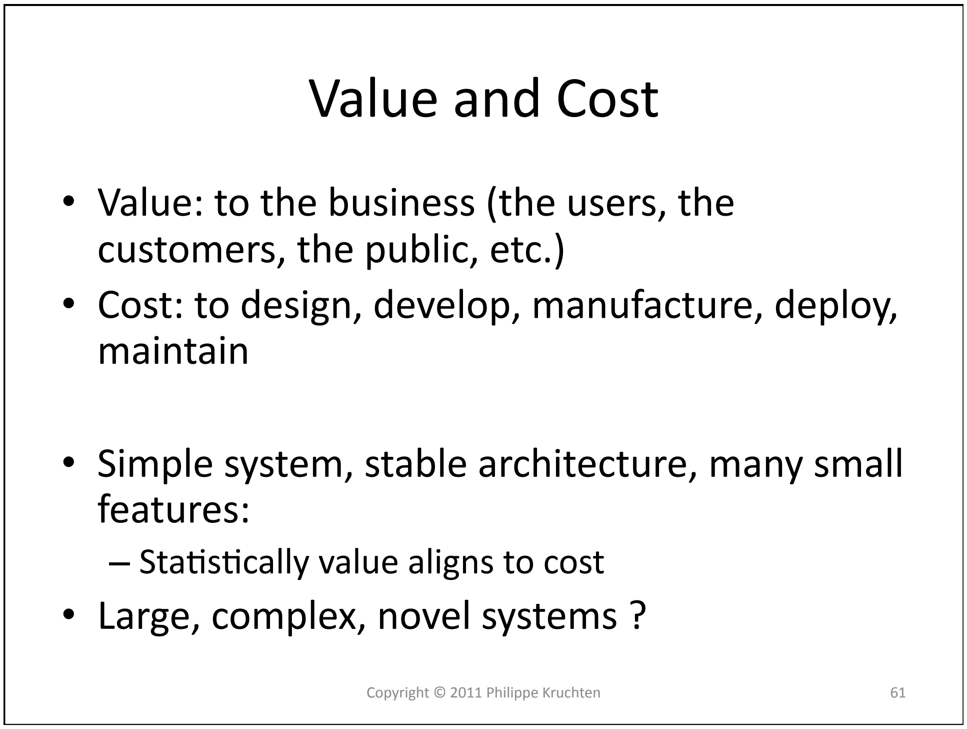
- There are no fixed prescriptions for systematically deriving architecture from requirements; there are only guidelines.
- Architecture designs can be reviewed.
- Architectural prototypes can be thoroughly tested.
- Iterative refinement is the only feasible approach to developing architectures for complex systems.

59

Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

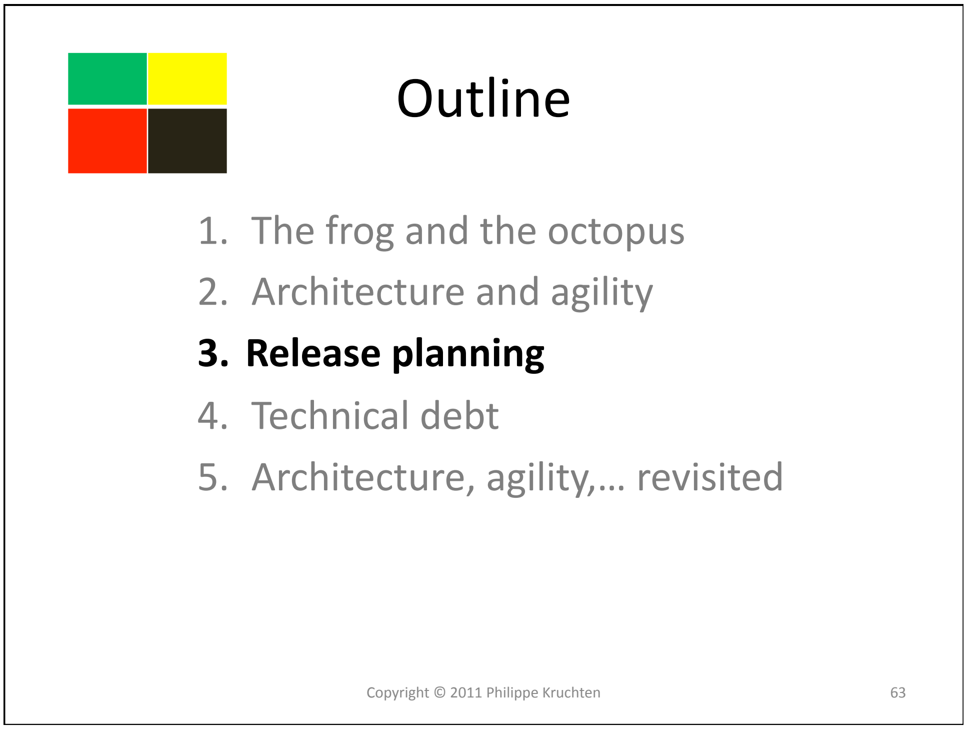
60



Value and Cost

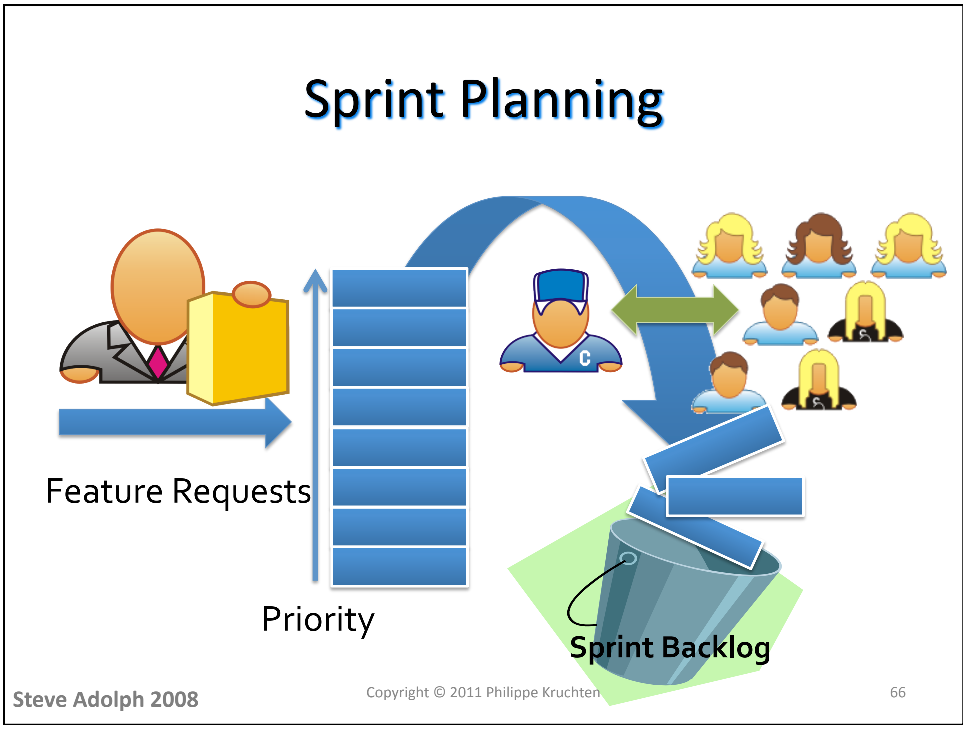
- Value: to the business (the users, the customers, the public, etc.)
- Cost: to design, develop, manufacture, deploy, maintain

- Simple system, stable architecture, many small features:
  - Statistically value aligns to cost
- Large, complex, novel systems ?



Outline

1. The frog and the octopus
2. Architecture and agility
3. **Release planning**
4. Technical debt
5. Architecture, agility,… revisited



Sprint Planning

Feature Requests

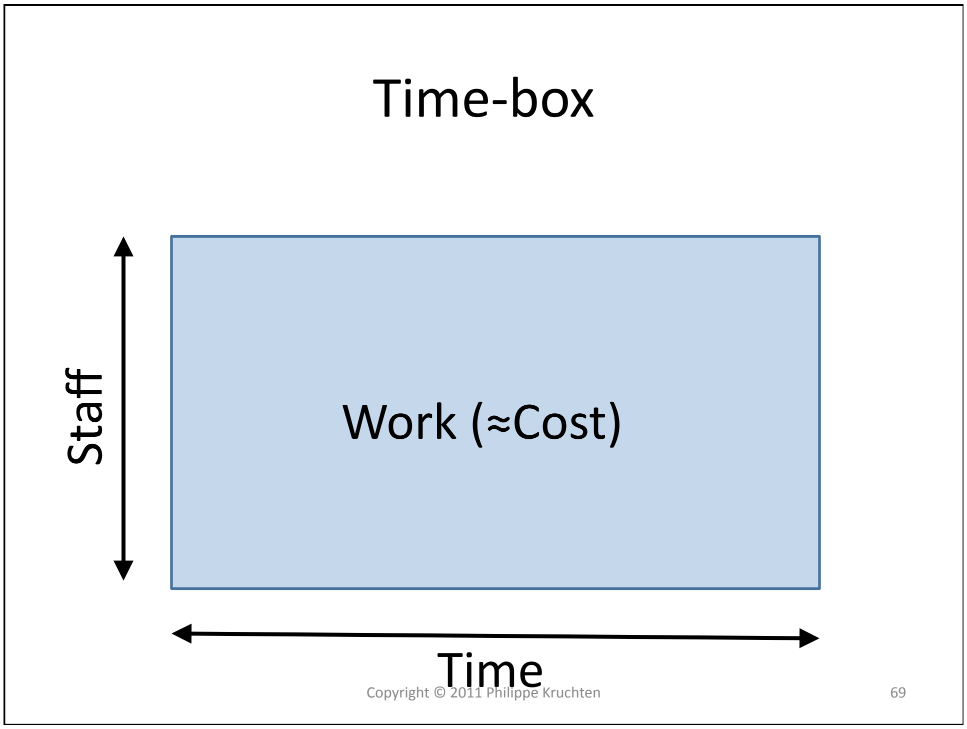
Priority

Sprint Backlog

Steve Adolph 2008

Copyright © 2011 Philippe Kruchten

66



Time-box

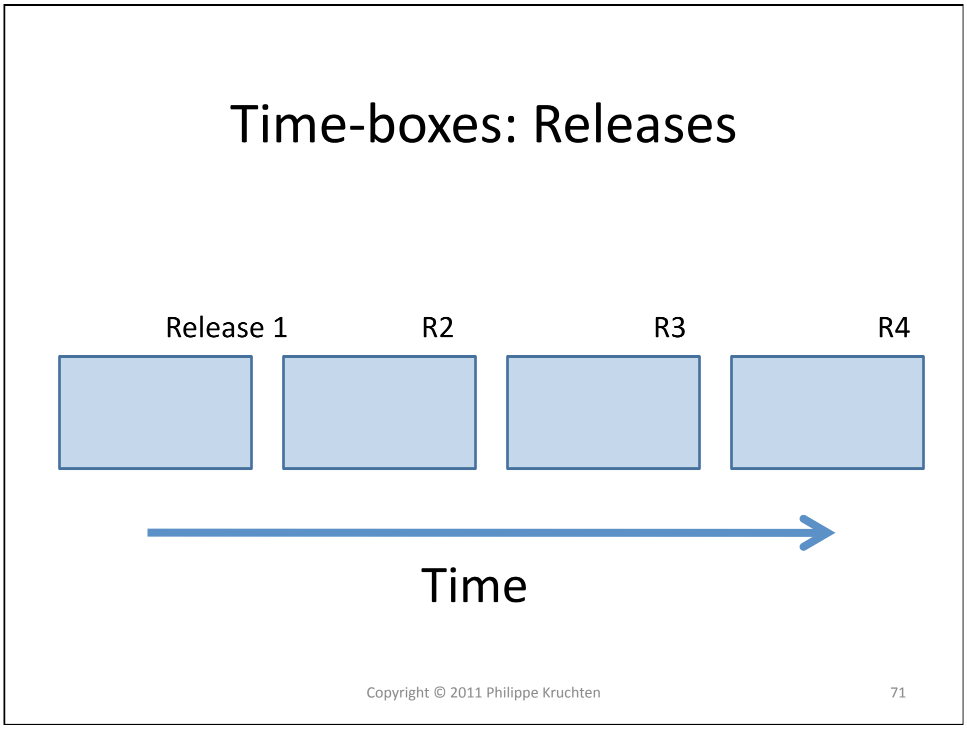
Staff

Work (≈Cost)

Time

Copyright © 2011 Philippe Kruchten

69

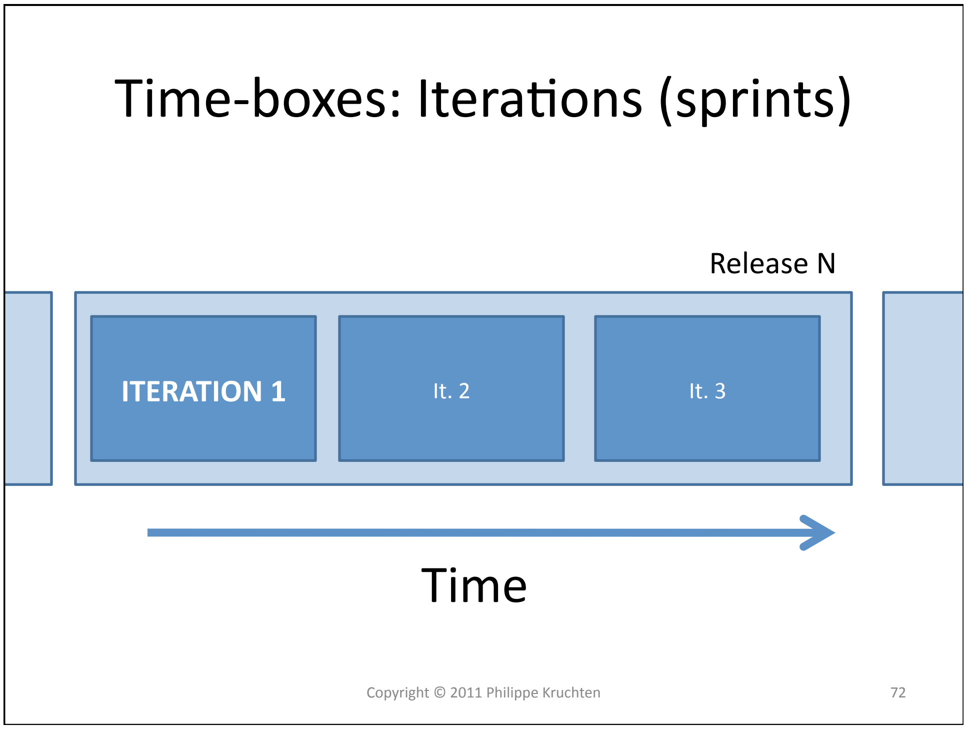


Time-boxes: Releases

Release 1 R2 R3 R4

Time

71



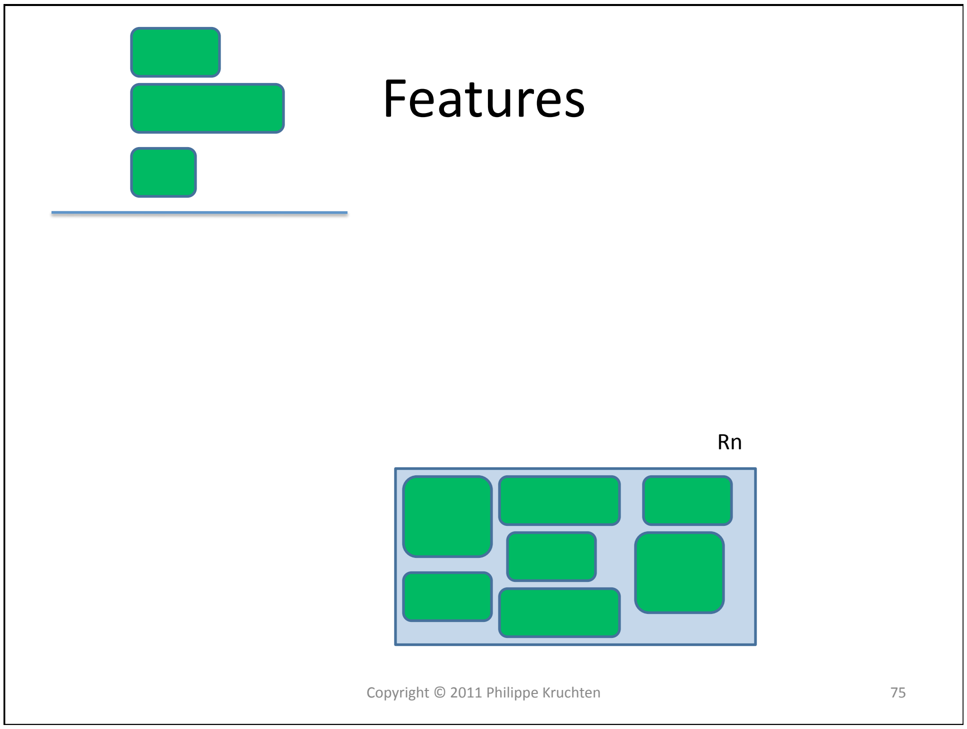
Time-boxes: Iterations (sprints)

Release N

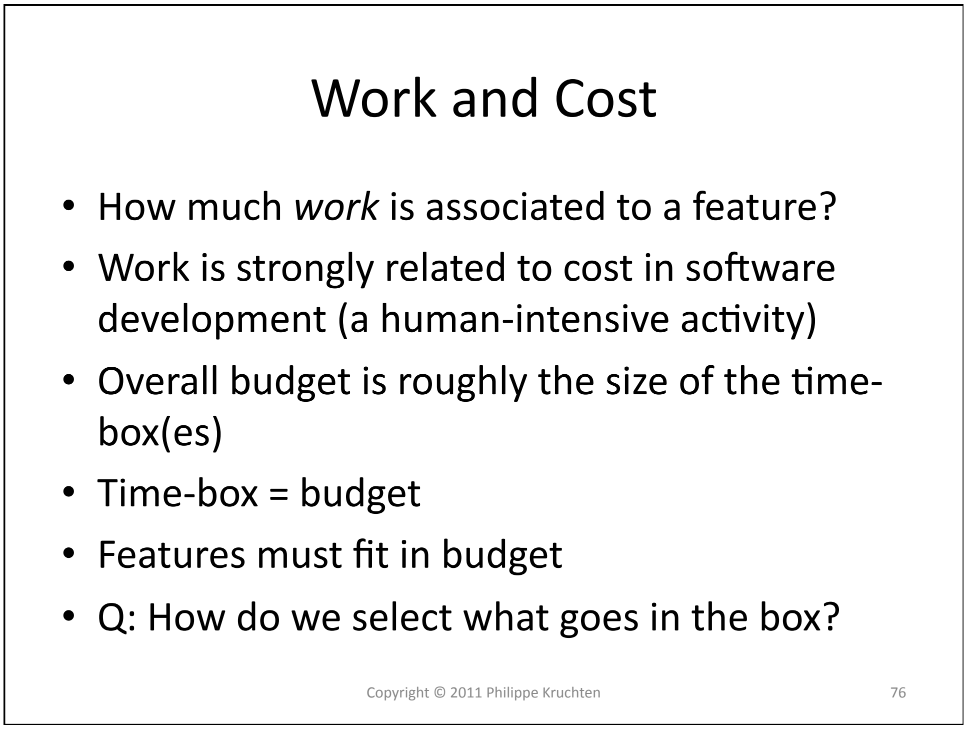
ITERATION 1 It. 2 It. 3

Time

72

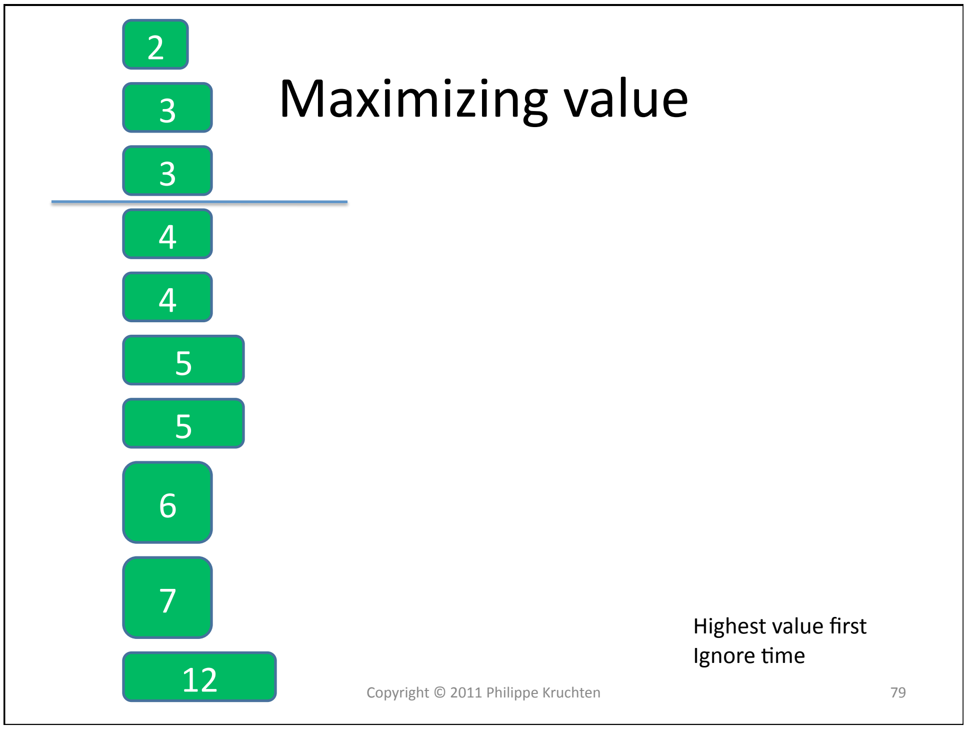


Features



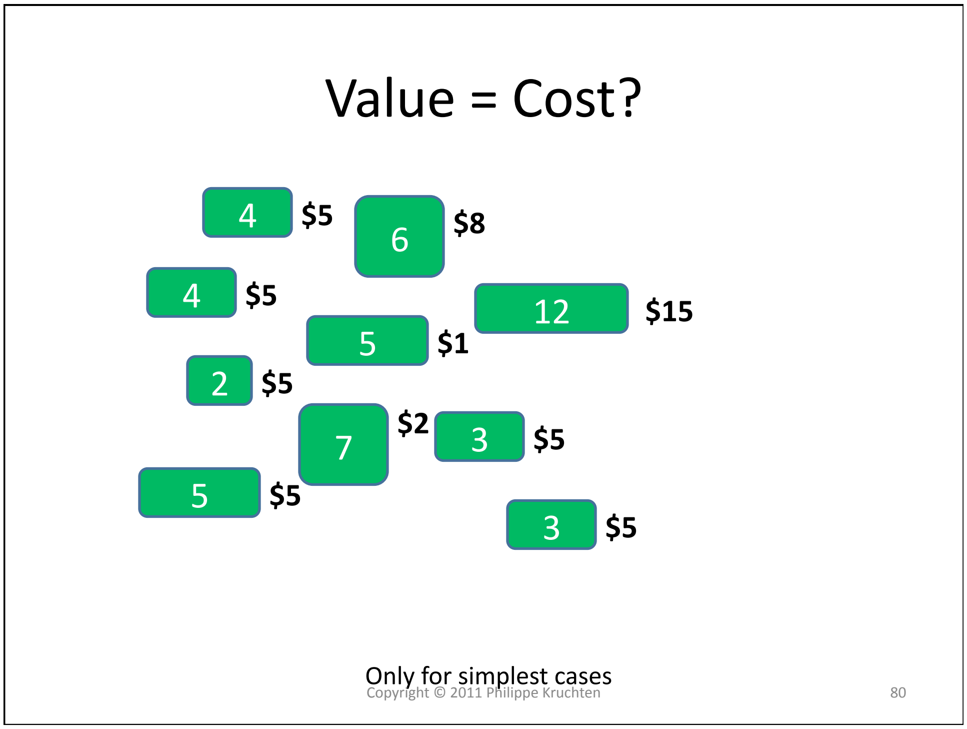
Work and Cost

- How much *work* is associated to a feature?
- Work is strongly related to cost in software development (a human-intensive activity)
- Overall budget is roughly the size of the time-box(es)
- Time-box = budget
- Features must fit in budget
- Q: How do we select what goes in the box?



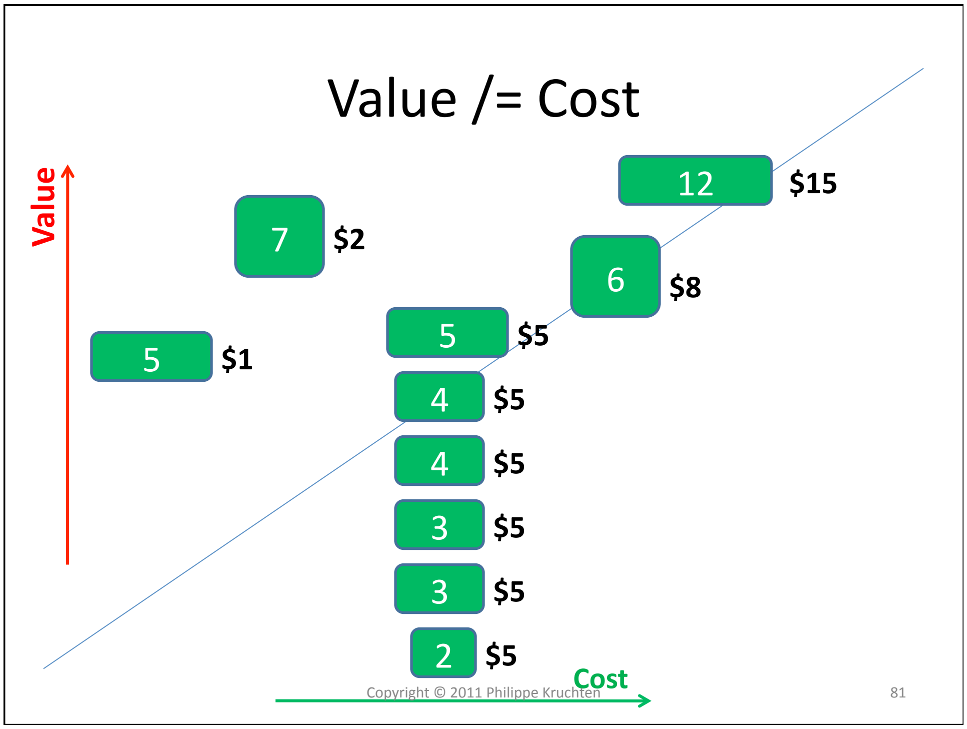
Maximizing value

Highest value first
Ignore time



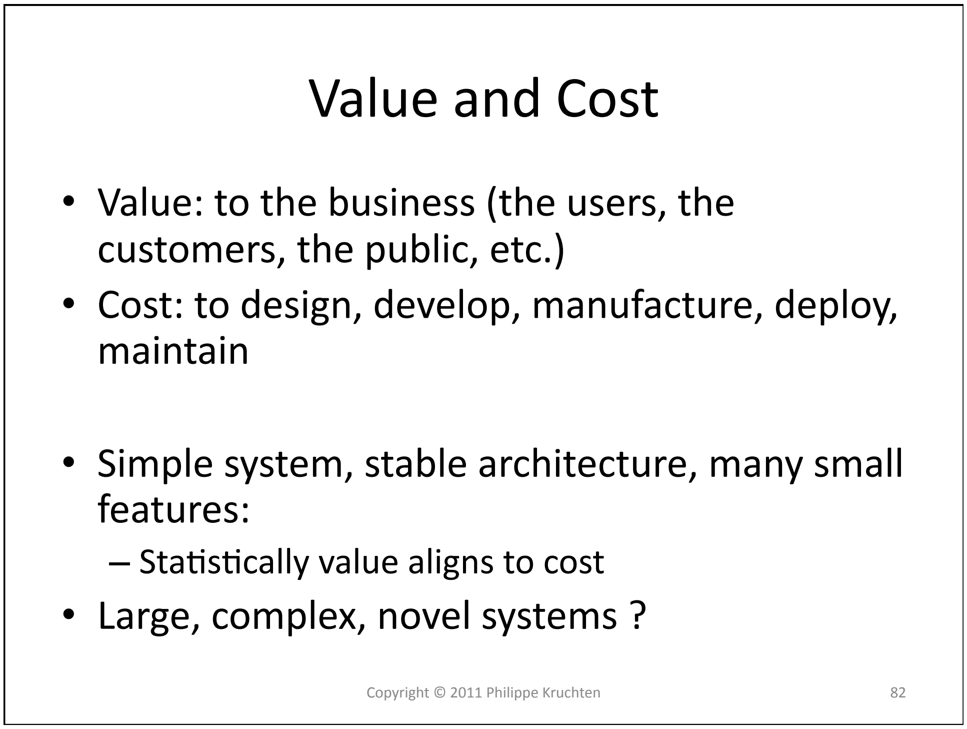
Value = Cost?

Only for simplest cases



Value /= Cost

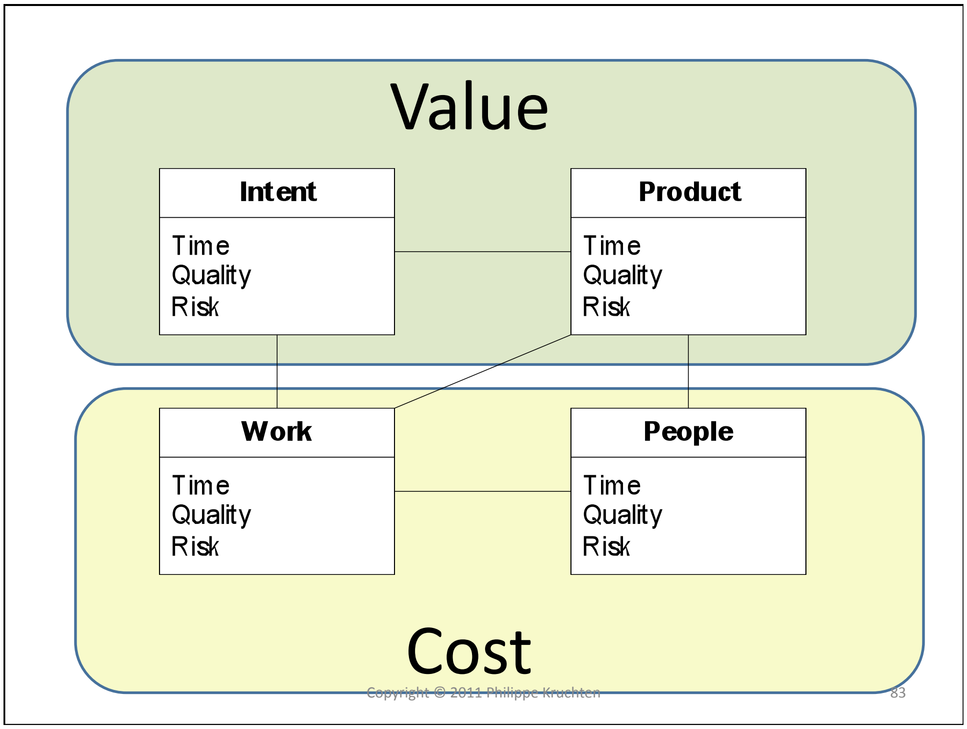
81



Value and Cost

- Value: to the business (the users, the customers, the public, etc.)
- Cost: to design, develop, manufacture, deploy, maintain
- Simple system, stable architecture, many small features:
  - Statistically value aligns to cost
- Large, complex, novel systems ?

82



Value

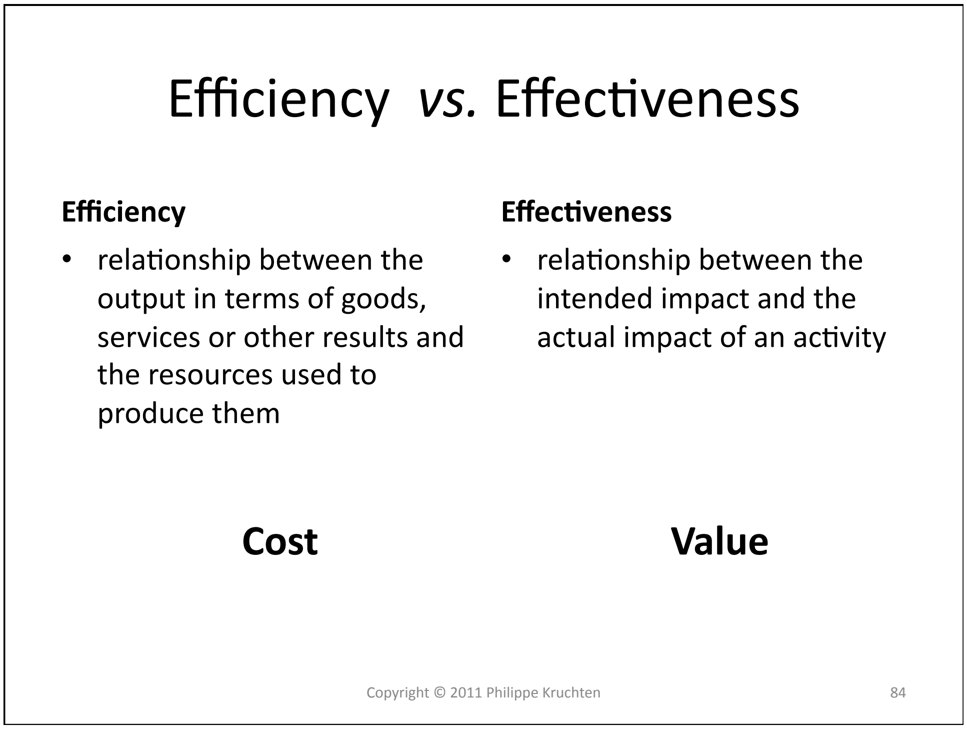
Intent
Time
Quality
Risk

Product
Time
Quality
Risk

Work
Time
Quality
Risk

People
Time
Quality
Risk

Cost



Efficiency *vs.* Effectiveness

Efficiency

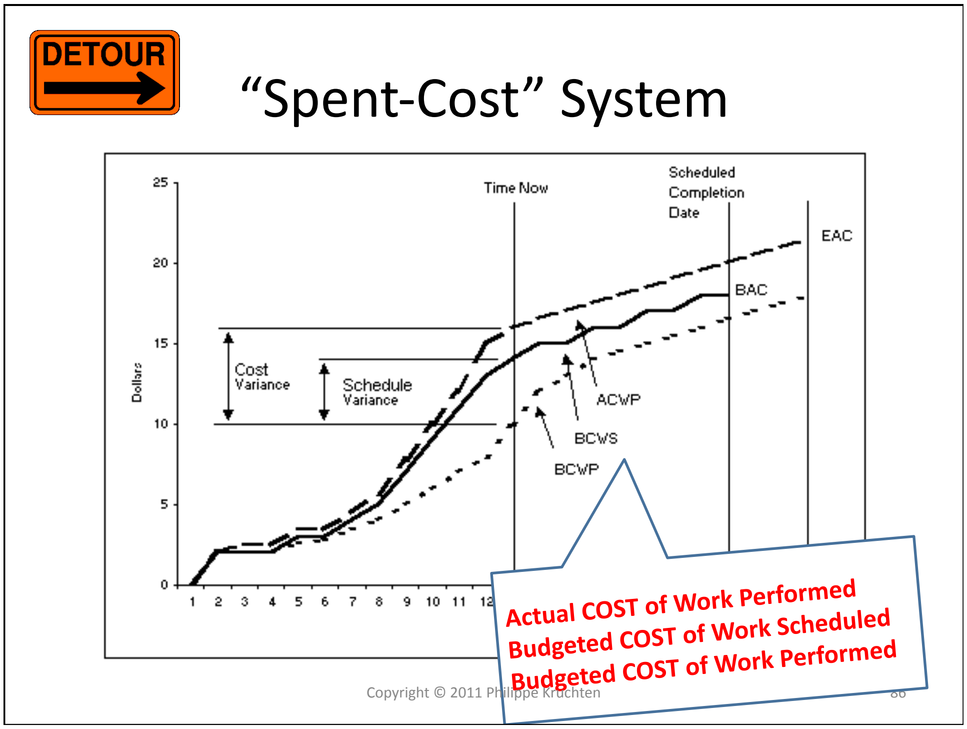
- relationship between the output in terms of goods, services or other results and the resources used to produce them

Effectiveness

- relationship between the intended impact and the actual impact of an activity

Cost

Value

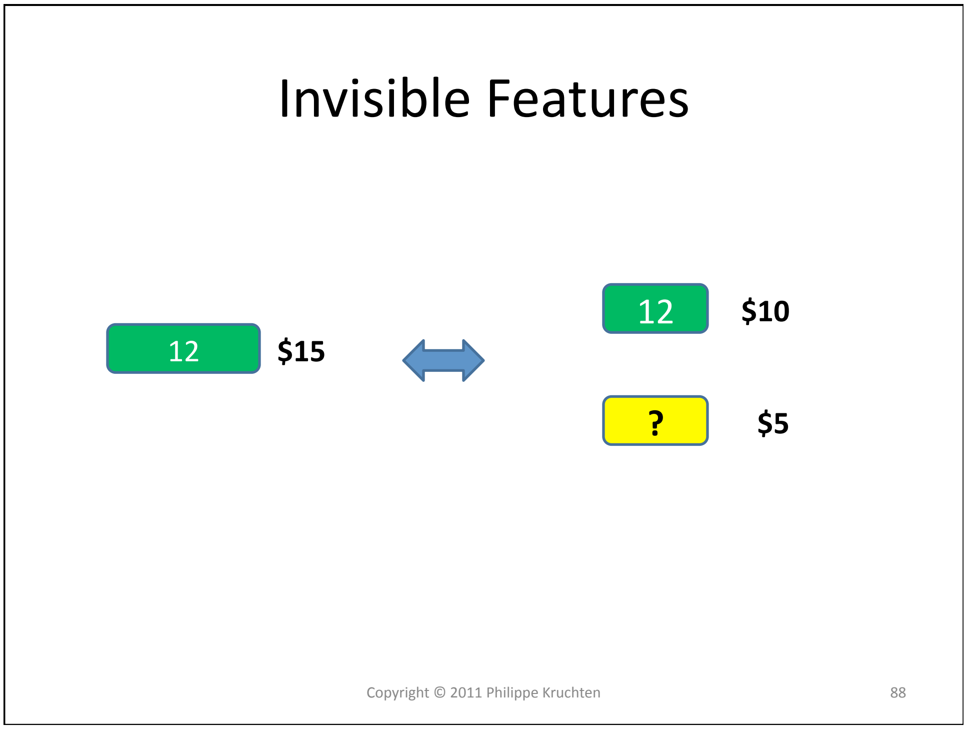


DETOUR

“Spent-Cost” System

Actual COST of Work Performed
Budgeted COST of Work Scheduled
Budgeted COST of Work Performed

Copyright © 2011 Philippe Kruchten 86

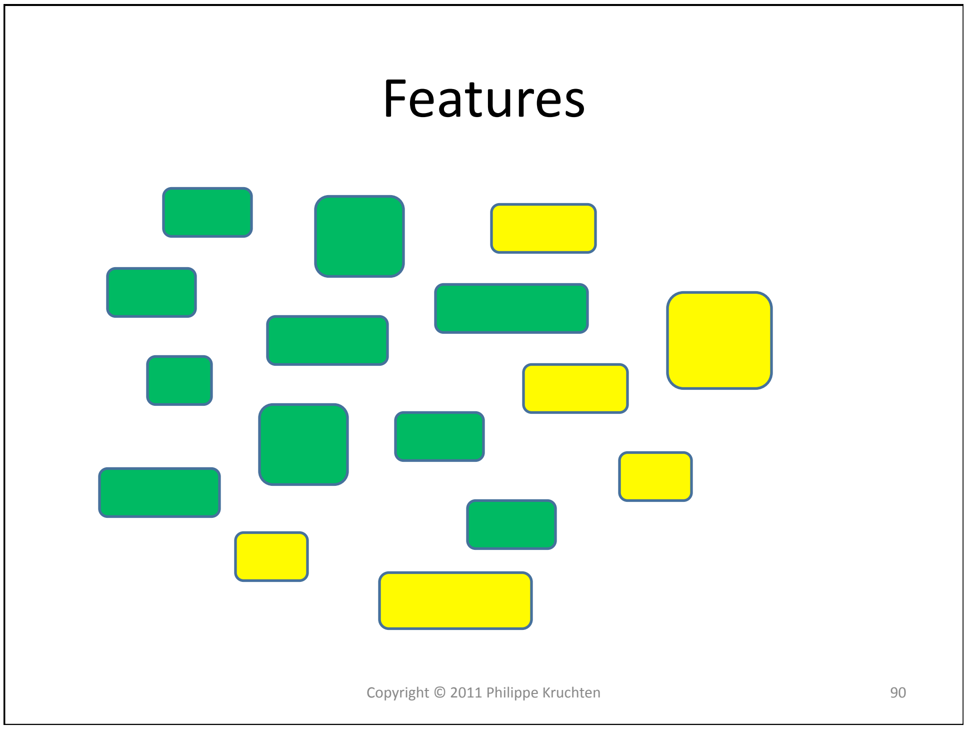


Invisible Features

Copyright © 2011 Philippe Kruchten 88

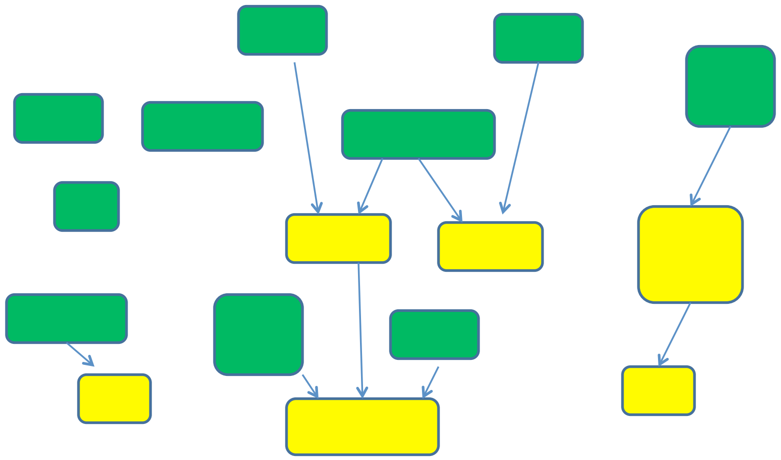
Invisible Features

- **Architecture**
- Infrastructure
- Common elements
- Framework
- Libraries
- Reuse
- DSL
- *Product line*



Features

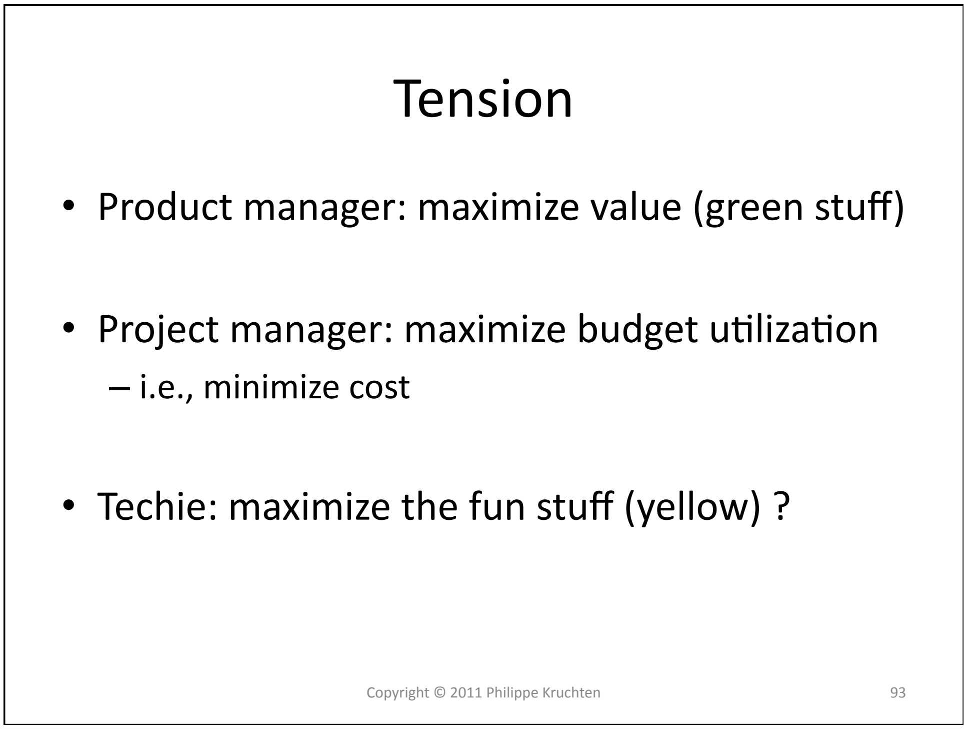
Dependencies



Release Planning

- Time-box = budget
- Fill the time-box with a combination of visible and invisible features
- … while maximizing value

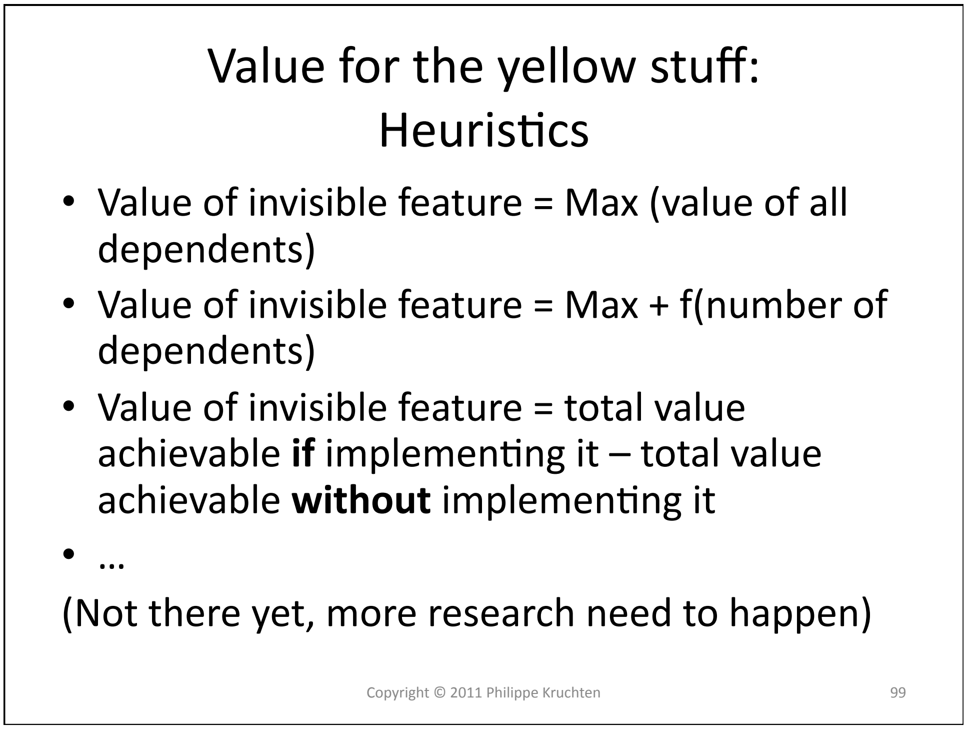
- Easy, no?



Tension

- Product manager: maximize value (green stuff)
- Project manager: maximize budget utilization
  - i.e., minimize cost
- Techie: maximize the fun stuff (yellow) ?

93



Value for the yellow stuff: Heuristics

- Value of invisible feature = Max (value of all dependents)
- Value of invisible feature = Max + f(number of dependents)
- Value of invisible feature = total value achievable **if** implementing it – total value achievable **without** implementing it
- …

(Not there yet, more research need to happen)

99

More on value & cost

- CBAM = Cost Benefit Analysis Method
  - Chap 12 in Bass, Clements, Kazman 2003
- IMF: Incremental Funding Method
  - Denne & Cleland-Huang, 2004

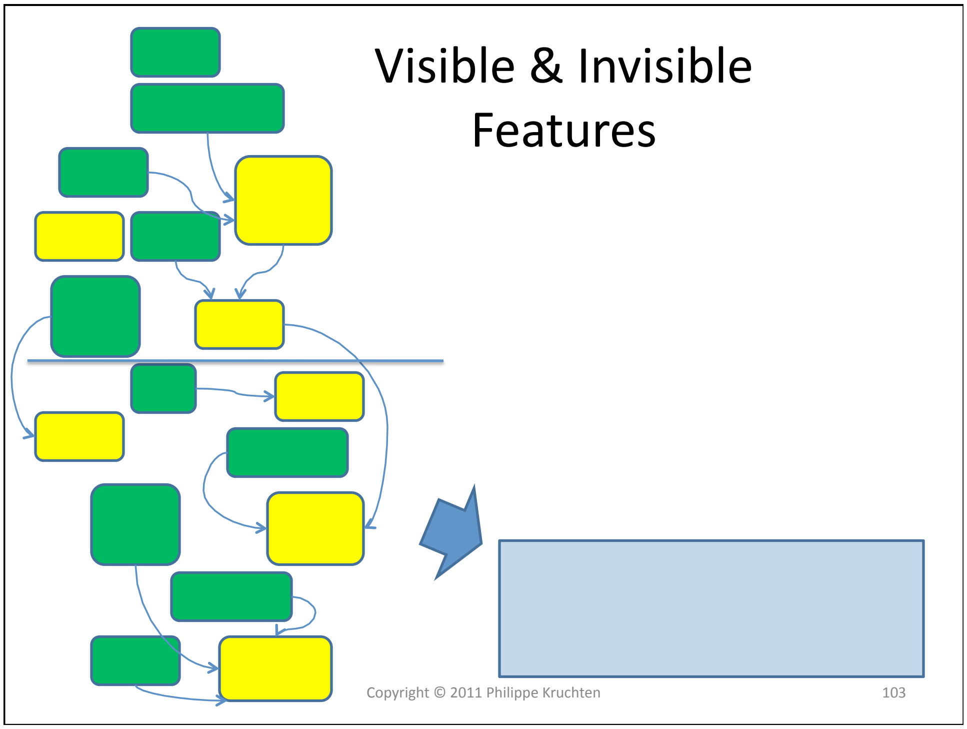
    Software by numbers

- Analytic Hierarchy Process (AHP) Saaty, 1990
- Evolve* - Hybrid
  - Günther Ruhe & D. Greer 2003, etc…

IFM: Incremental Funding Method

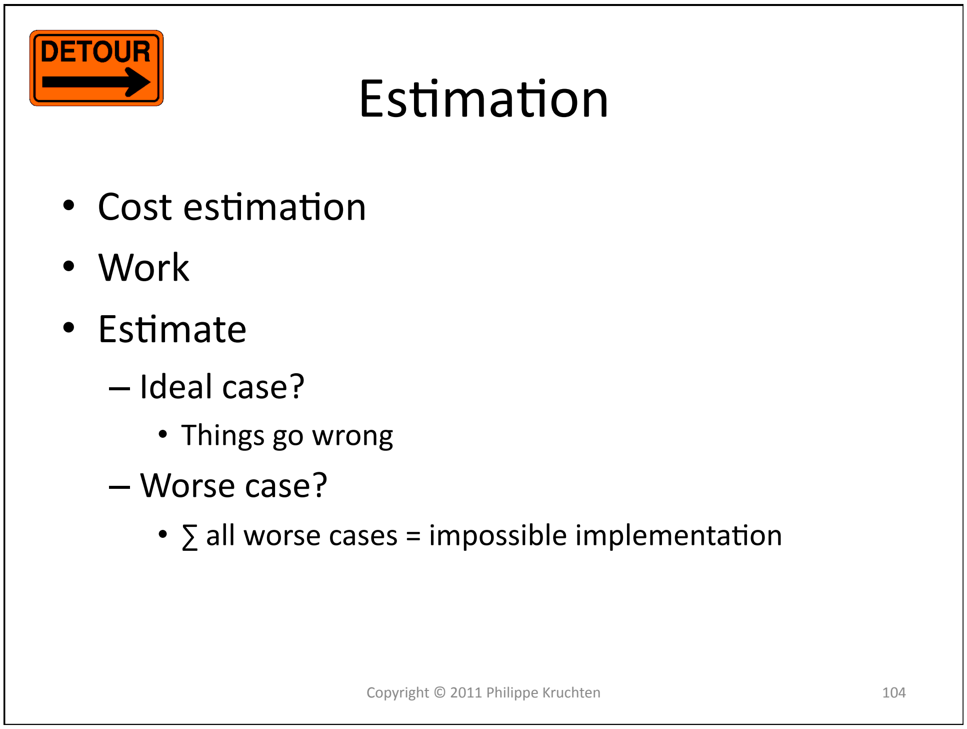
- MMF = Minimum Marketable Features
- AE = Architectural elements
- Cost
- MMF depends on AE
- Time and NPV = Net Present Value
- Strands = Sequences of dependent MMFs
- Heuristic

Denne & Huang, www.softwarebynumbers.org



Visible & Invisible Features

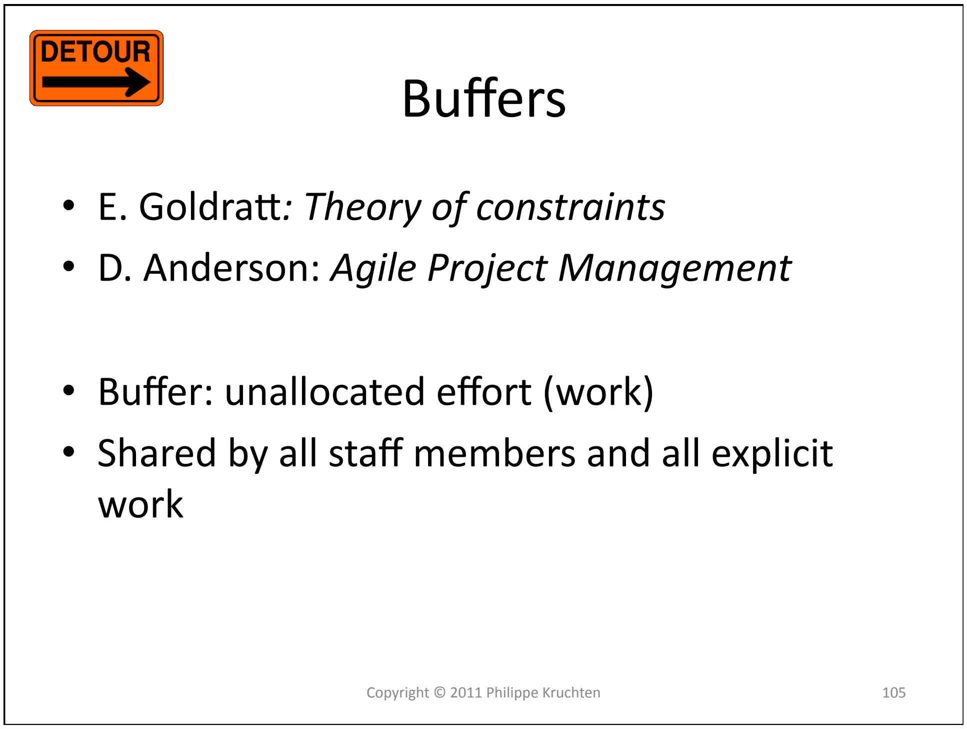
103



Estimation

- Cost estimation
- Work
- Estimate
  - Ideal case?
    - Things go wrong
  - Worse case?
    - ∑ all worse cases = impossible implementation

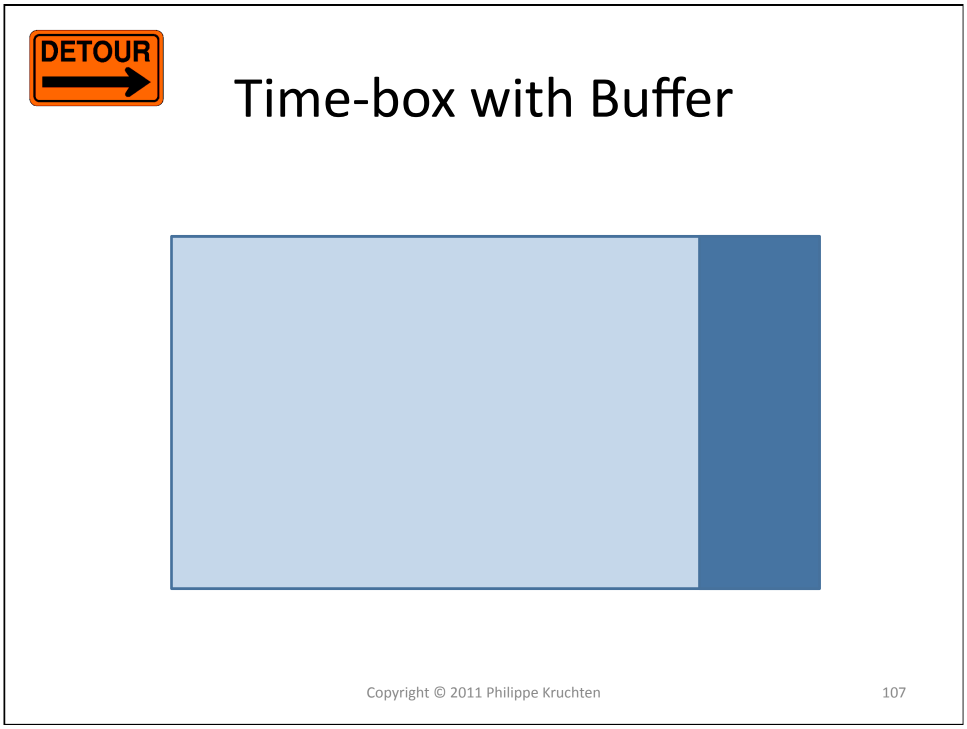
104



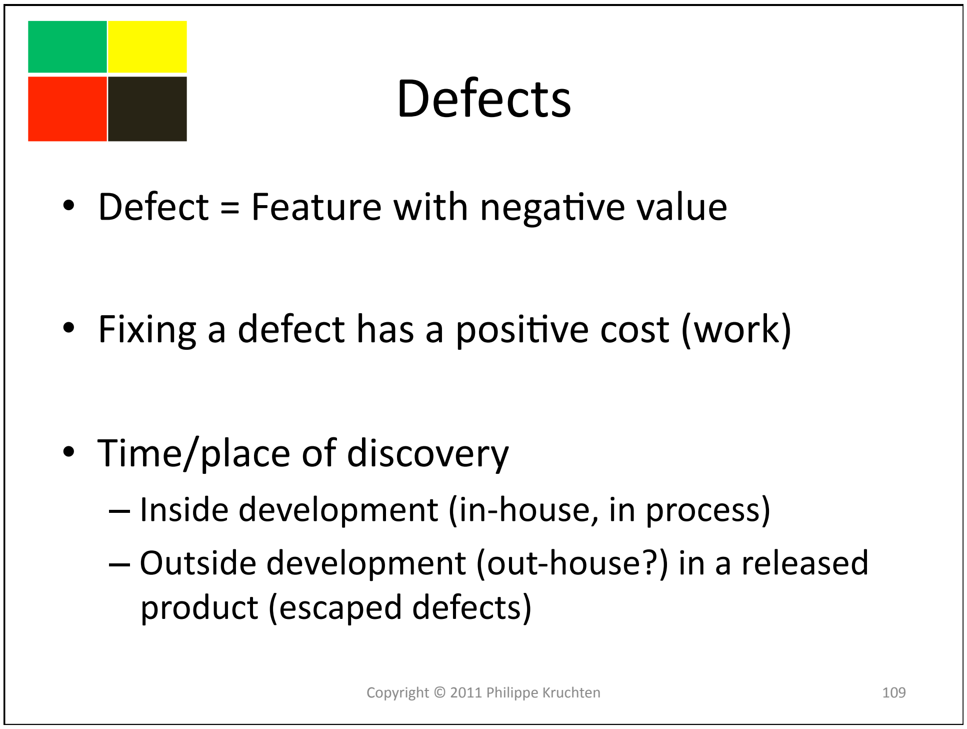
Buffers

- E. Goldratt*: Theory of constraints*
- D. Anderson: *Agile Project Management*

- Buffer: unallocated effort (work)
- Shared by all staff members and all explicit work



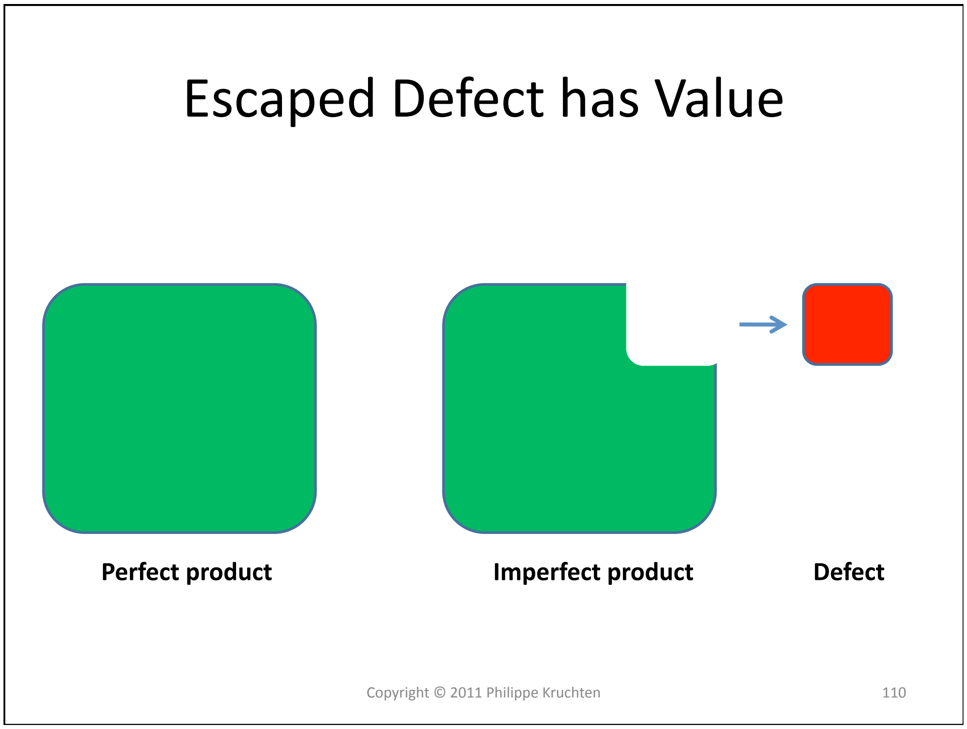
Time-box with Buffer



Defects

- Defect = Feature with negative value
- Fixing a defect has a positive cost (work)
- Time/place of discovery
  - Inside development (in-house, in process)
  - Outside development (out-house?) in a released product (escaped defects)

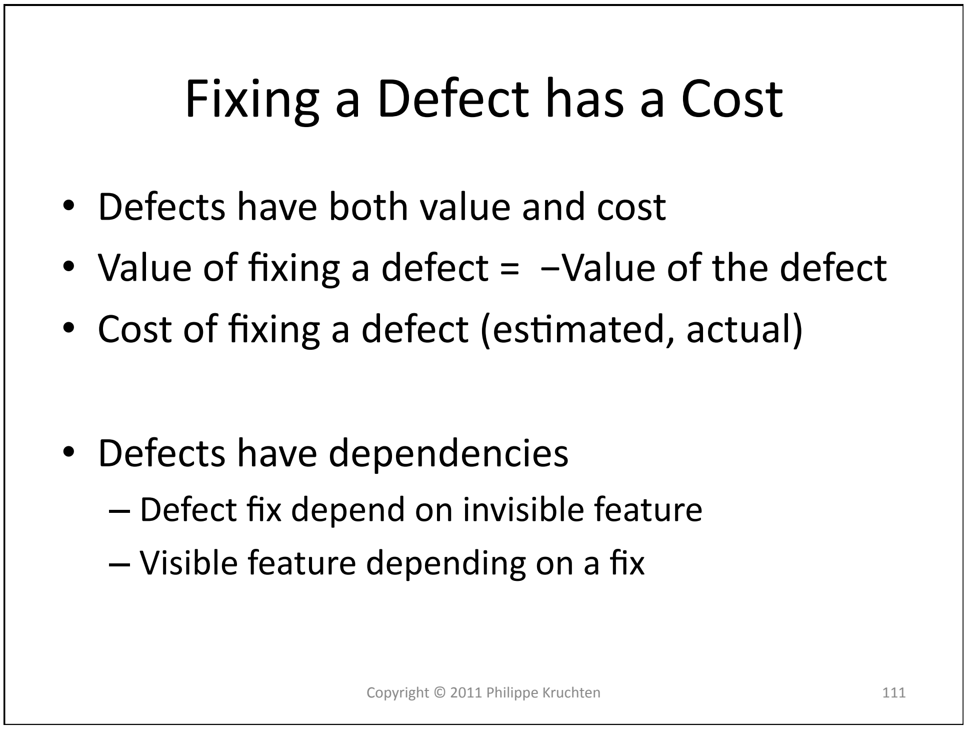
Copyright © 2011 Philippe Kruchten 109



Escaped Defect has Value

Perfect product **Imperfect product** **Defect**

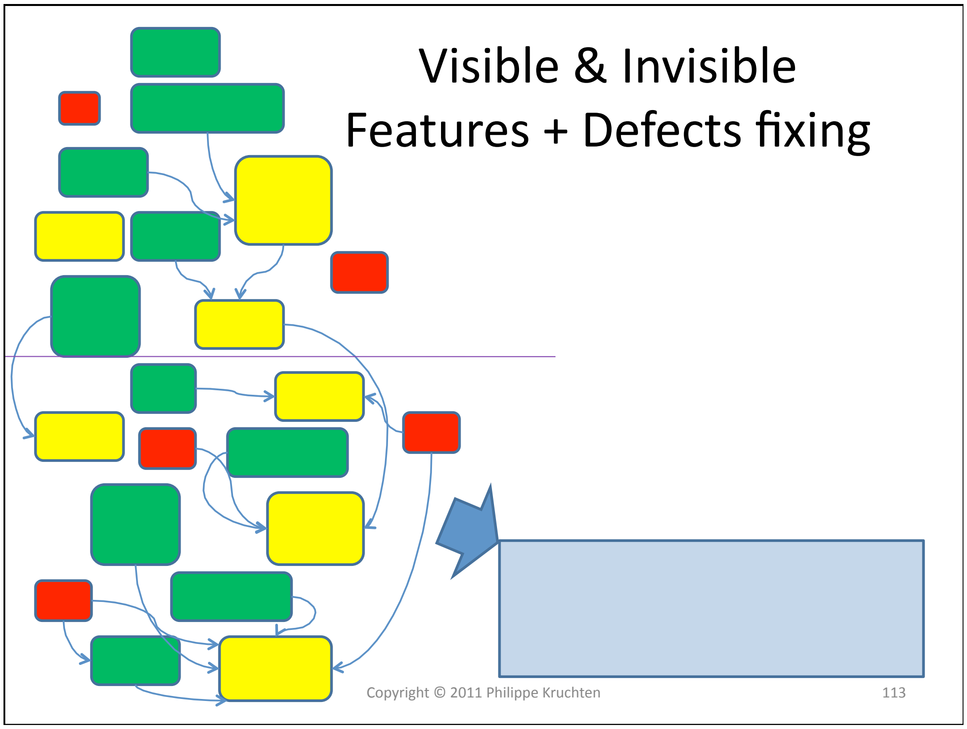
Copyright © 2011 Philippe Kruchten 110



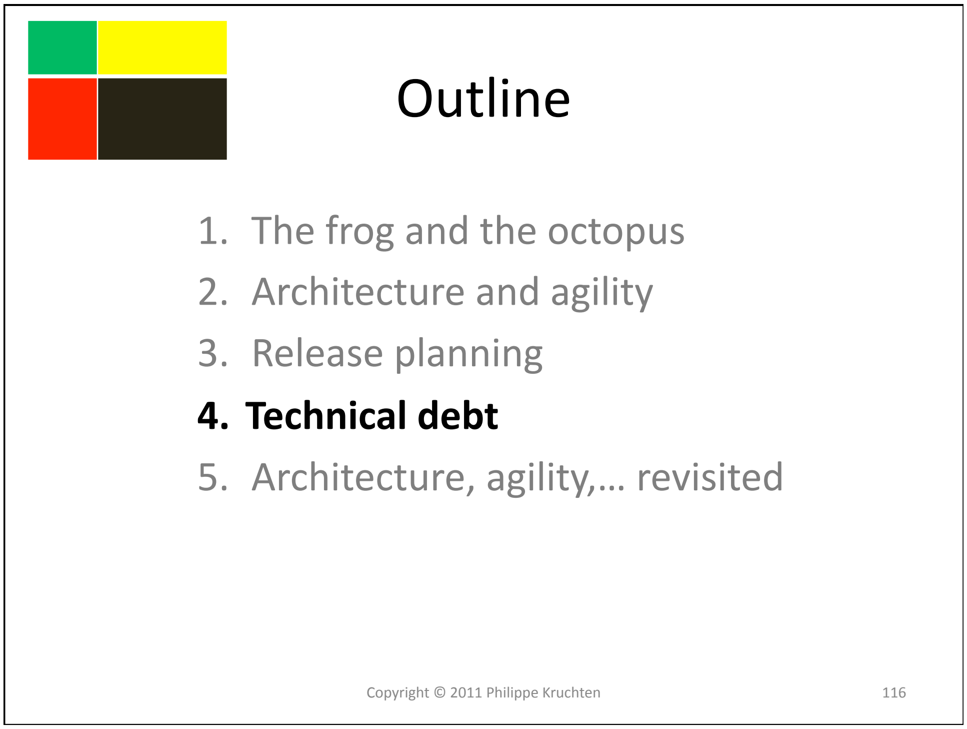
Fixing a Defect has a Cost

- Defects have both value and cost
- Value of fixing a defect = −Value of the defect
- Cost of fixing a defect (estimated, actual)

- Defects have dependencies
  - Defect fix depend on invisible feature
  - Visible feature depending on a fix

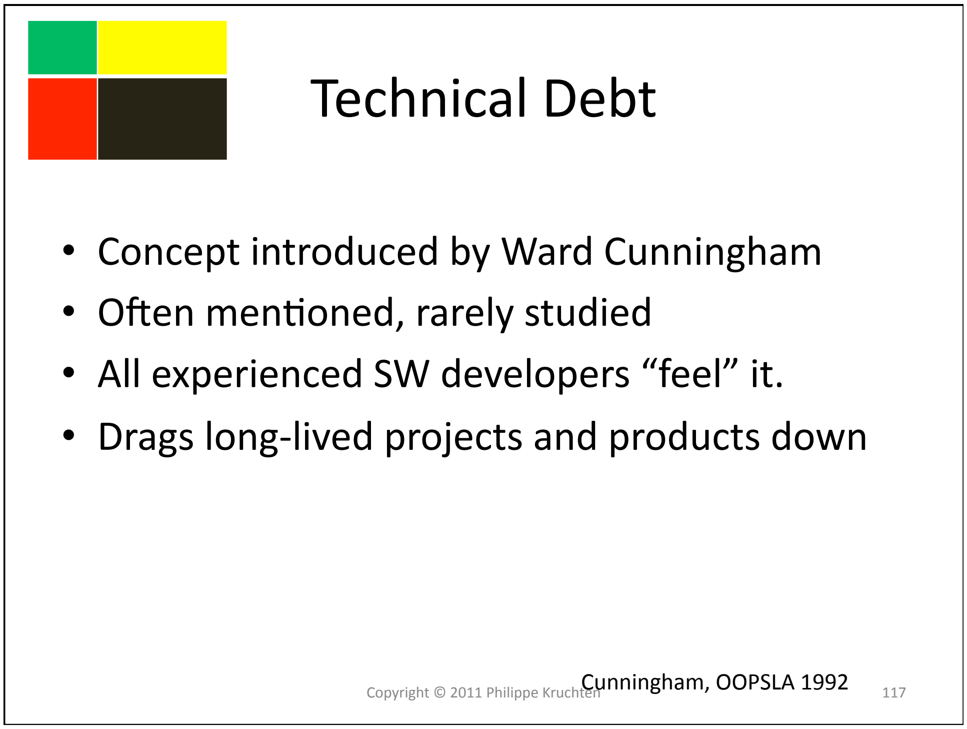


Visible & Invisible Features + Defects fixing



Outline

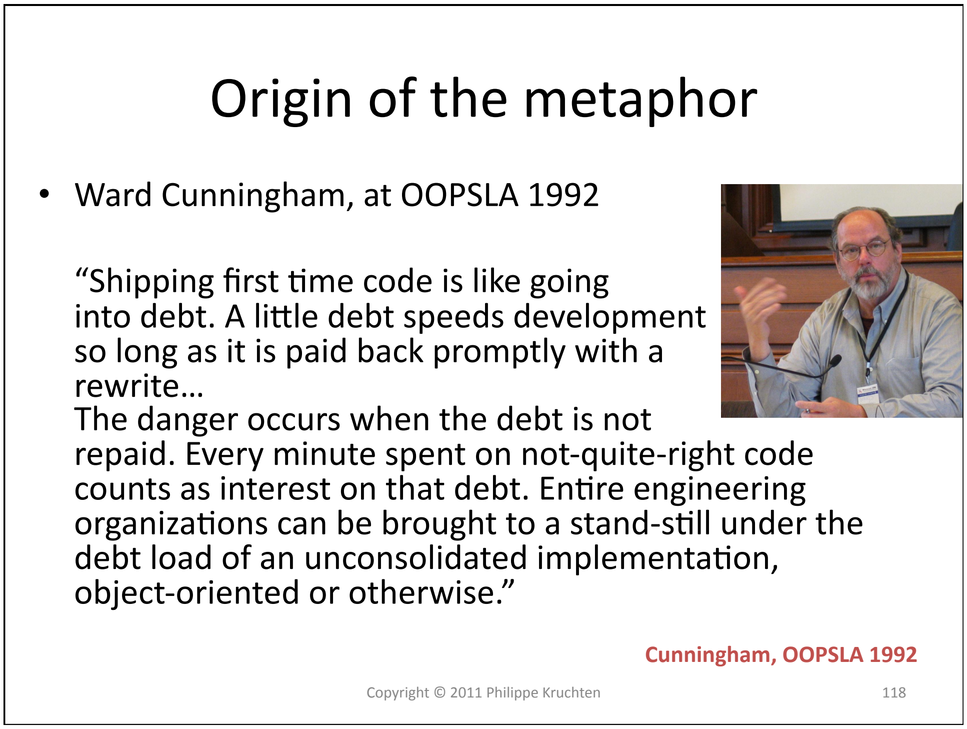
1. The frog and the octopus
2. Architecture and agility
3. Release planning
4. **Technical debt**
5. Architecture, agility,… revisited



Technical Debt

- Concept introduced by Ward Cunningham
- Often mentioned, rarely studied
- All experienced SW developers “feel” it.
- Drags long-lived projects and products down

Cunningham, OOPSLA 1992

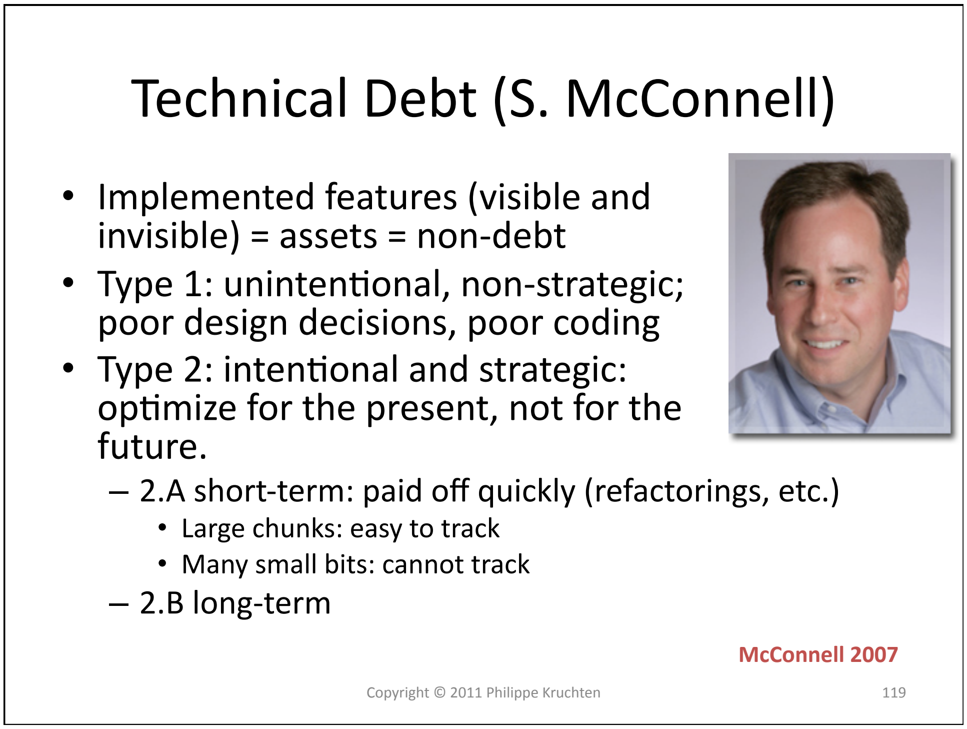


Origin of the metaphor

- Ward Cunningham, at OOPSLA 1992

  "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite…
  The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."

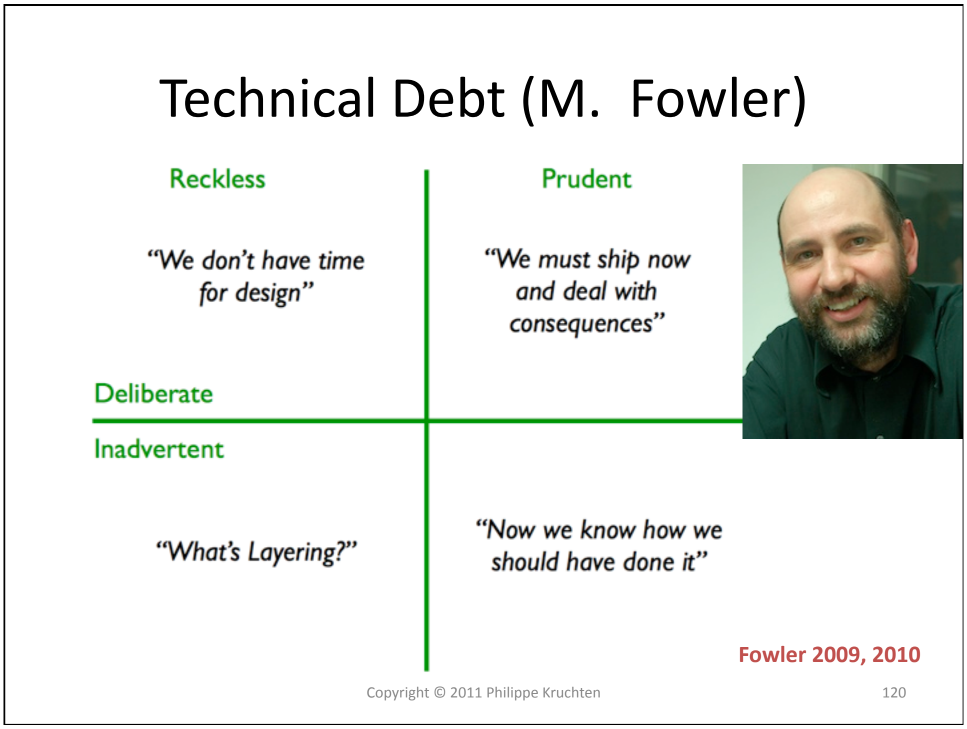
Cunningham, OOPSLA 1992



Technical Debt (S. McConnell)

- Implemented features (visible and invisible) = assets = non-debt
- Type 1: unintentional, non-strategic; poor design decisions, poor coding
- Type 2: intentional and strategic: optimize for the present, not for the future.
  - 2.A short-term: paid off quickly (refactorings, etc.)
    - Large chunks: easy to track
    - Many small bits: cannot track
  - 2.B long-term

McConnell 2007



Technical Debt (M. Fowler)

Reckless

Prudent

"We don't have time for design"

"We must ship now and deal with consequences"

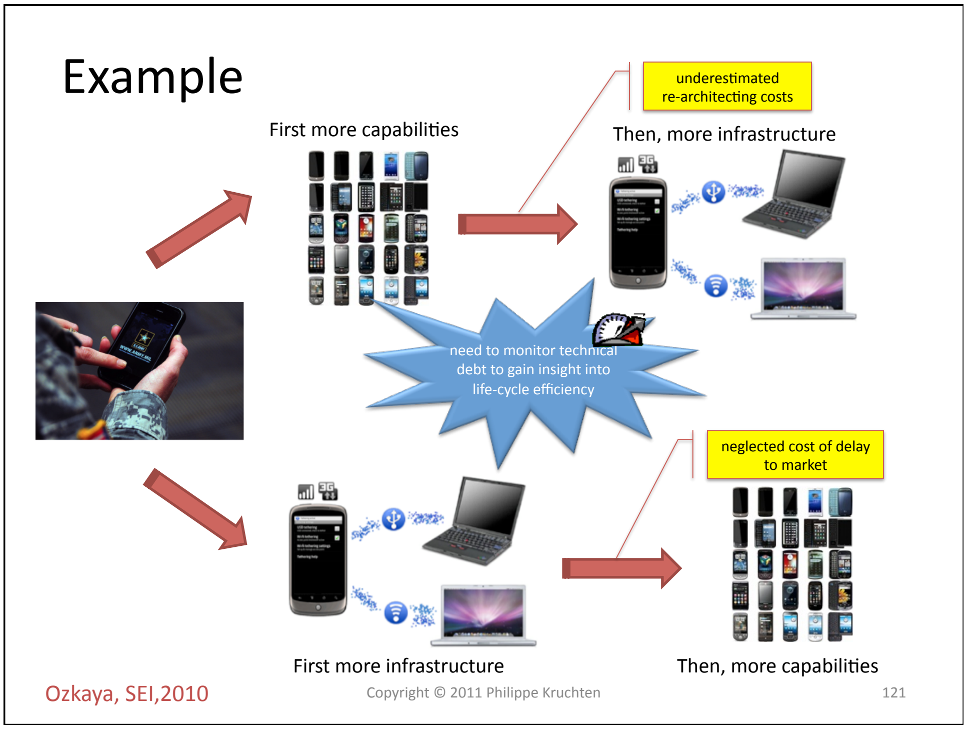
Deliberate

Inadvertent

"What's Layering?"

"Now we know how we should have done it"

Fowler 2009, 2010



Example

First more capabilities

underestimated re-architecting costs

Then, more infrastructure

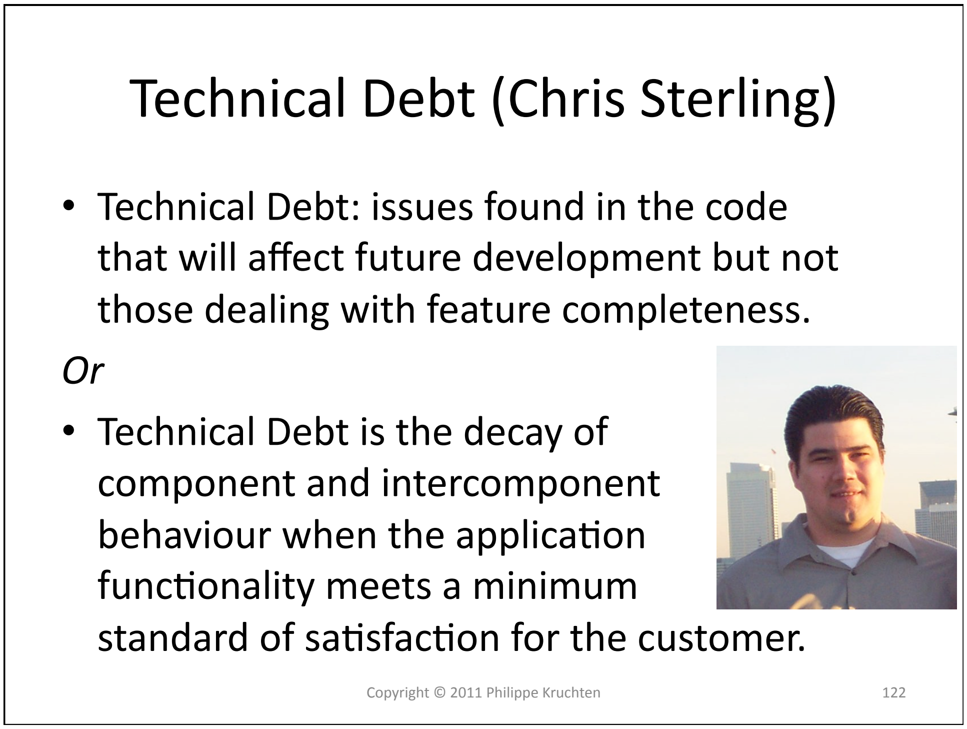
need to monitor technical debt to gain insight into life-cycle efficiency

neglected cost of delay to market

First more infrastructure

Then, more capabilities

Ozkaya, SEI,2010

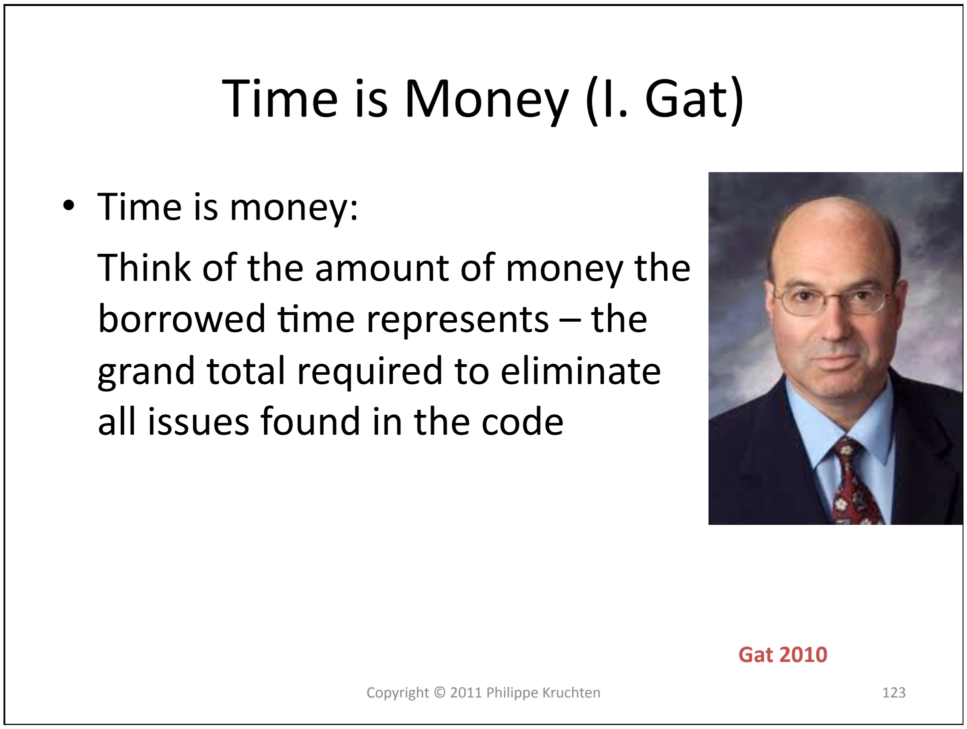


Technical Debt (Chris Sterling)

- Technical Debt: issues found in the code that will affect future development but not those dealing with feature completeness.

Or

- Technical Debt is the decay of component and intercomponent behaviour when the application functionality meets a minimum standard of satisfaction for the customer.

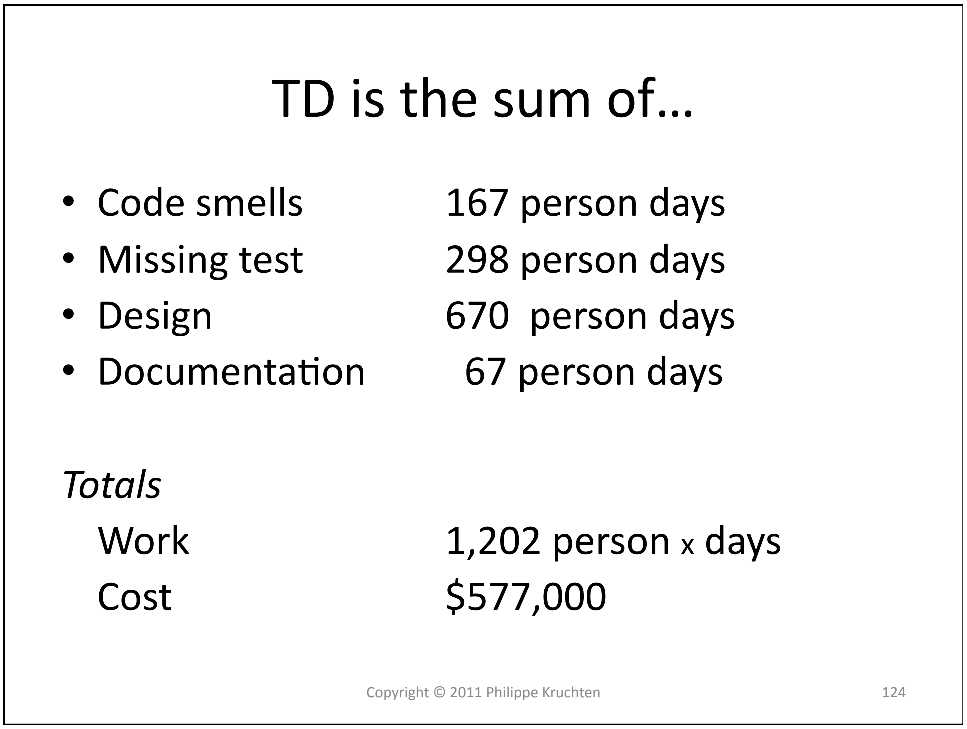


Time is Money (I. Gat)

- Time is money:

  Think of the amount of money the borrowed time represents – the grand total required to eliminate all issues found in the code

Gat 2010

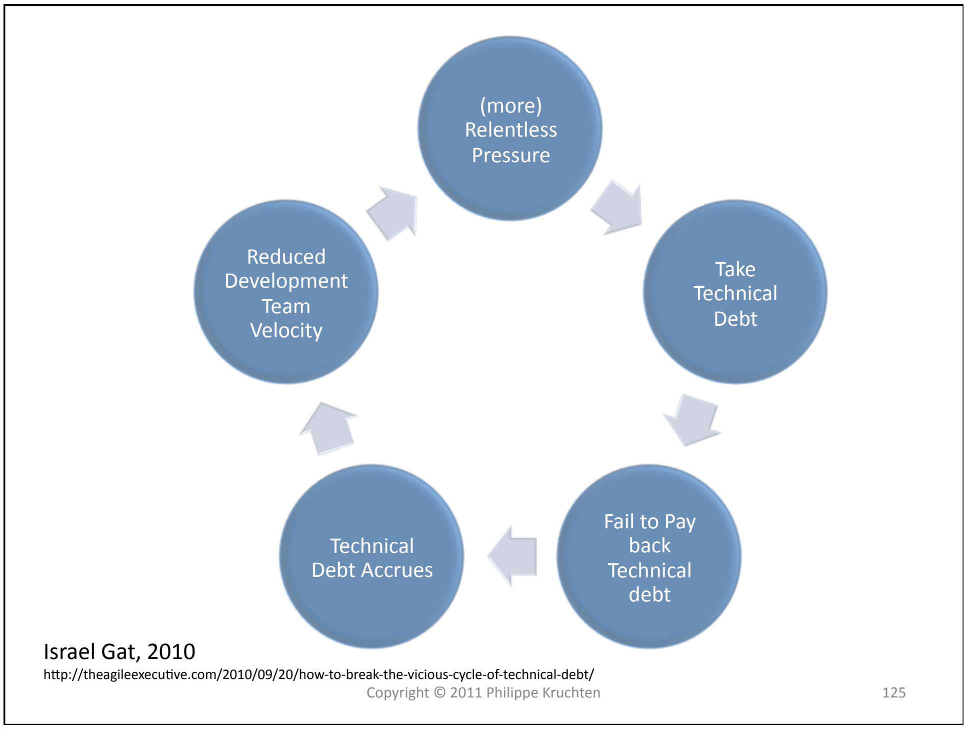


TD is the sum of…

• Code smells	167 person days
• Missing test	298 person days
• Design	670 person days
• Documentation	67 person days

Totals

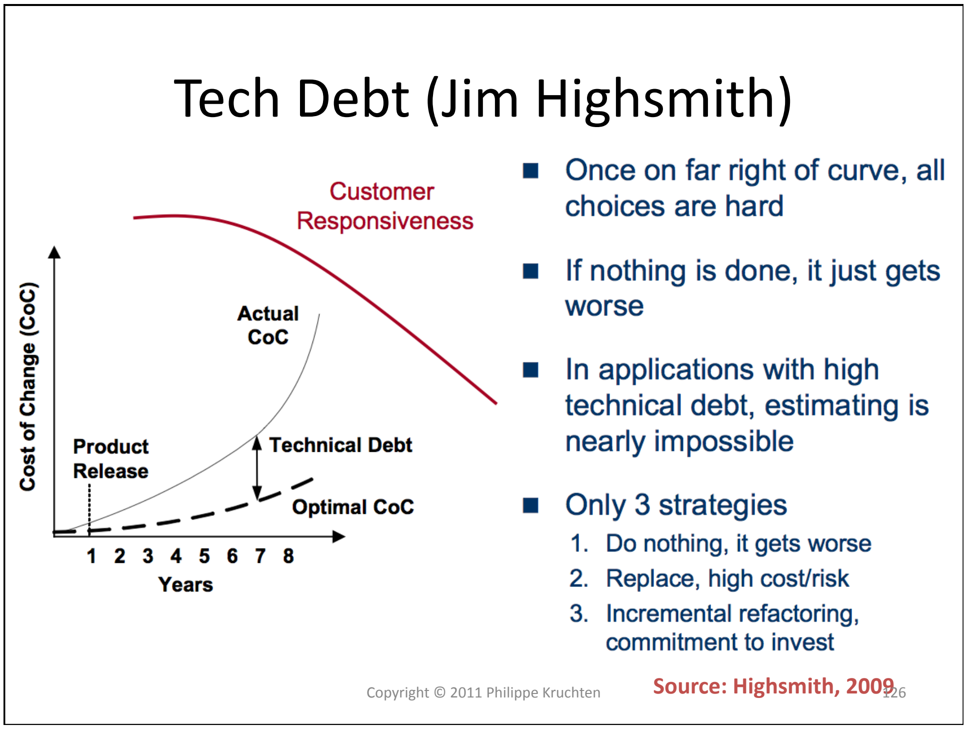
Work	1,202 person x days
Cost	$577,000



(more) Relentless Pressure
Take Technical Debt
Fail to Pay back Technical debt
Technical Debt Accrues
Reduced Development Team Velocity

Israel Gat, 2010

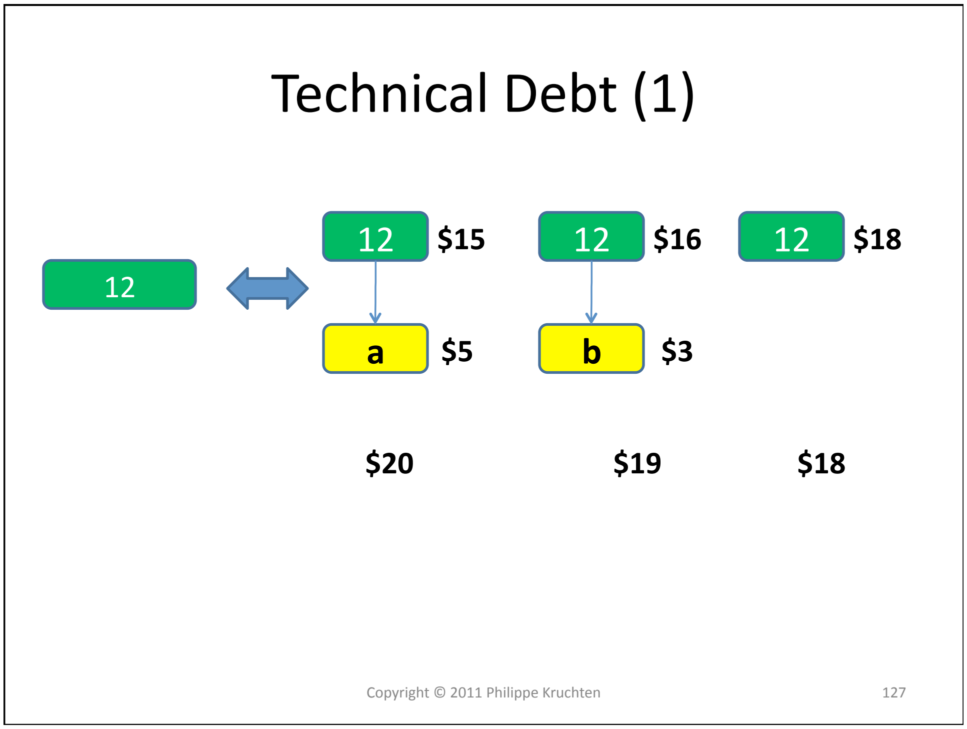
http://theagileexecutive.com/2010/09/20/how-to-break-the-vicious-cycle-of-technical-debt/



Tech Debt (Jim Highsmith)

- Once on far right of curve, all choices are hard
- If nothing is done, it just gets worse
- In applications with high technical debt, estimating is nearly impossible
- Only 3 strategies
  1. Do nothing, it gets worse
  2. Replace, high cost/risk
  3. Incremental refactoring, commitment to invest

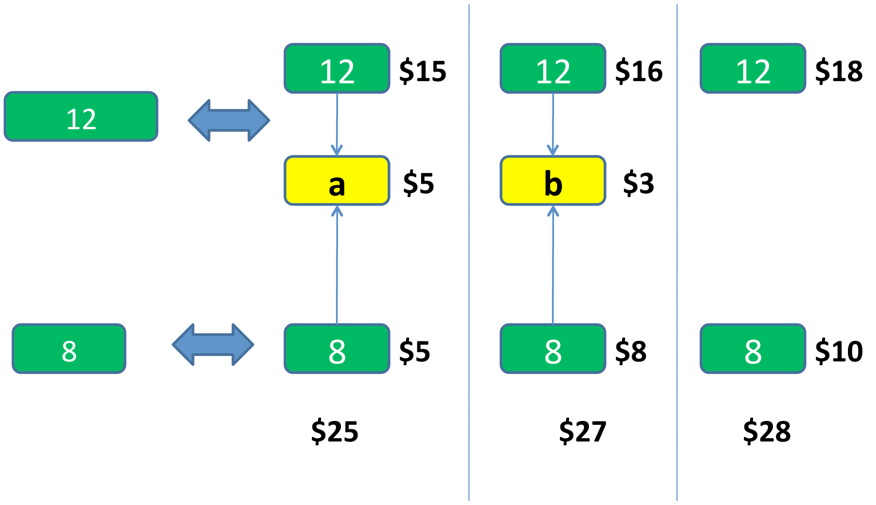
Source: Highsmith, 2009



Technical Debt (1)

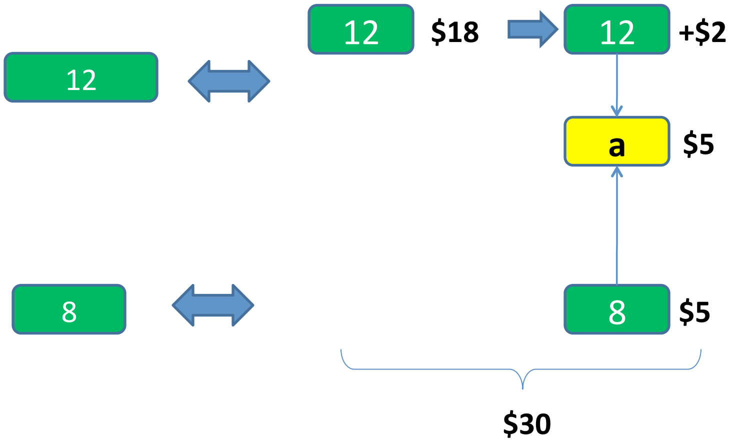
$20 $19 $18

Technical Debt (2)

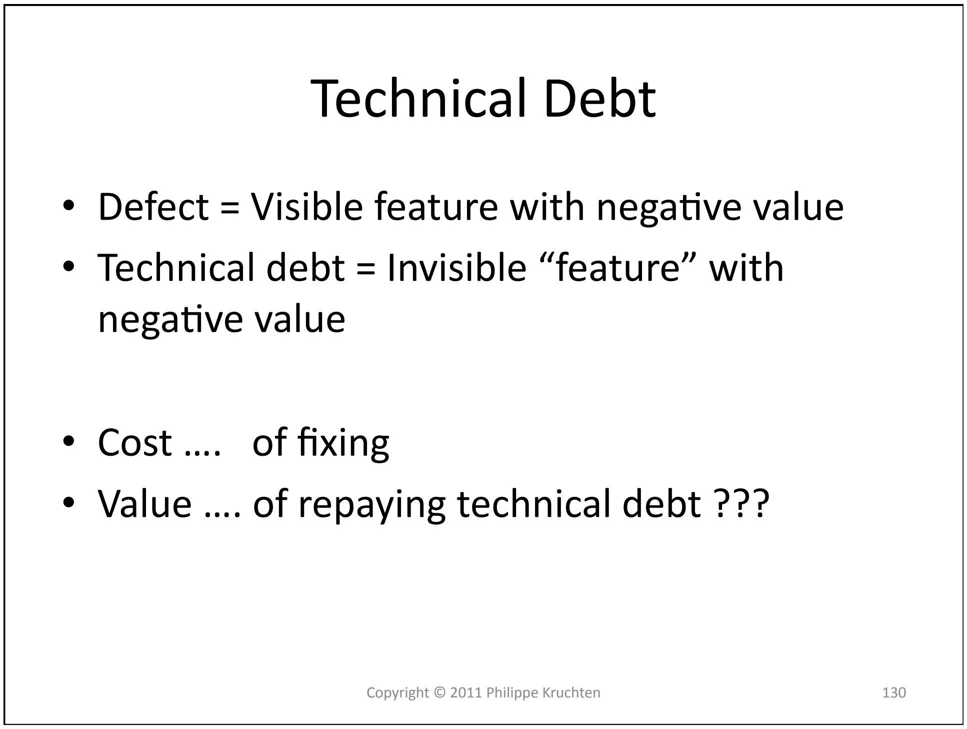


128

Technical Debt (3)



129

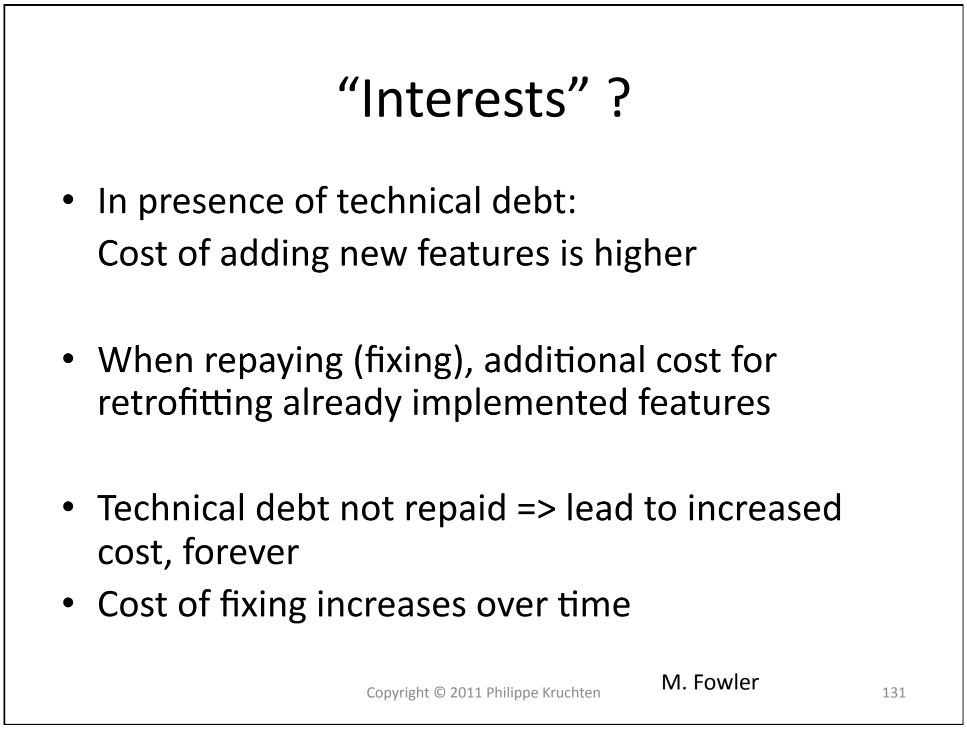


Technical Debt

- Defect = Visible feature with negative value
- Technical debt = Invisible "feature" with negative value

- Cost …. of fixing
- Value …. of repaying technical debt ???

Copyright © 2011 Philippe Kruchten 130



"Interests" ?

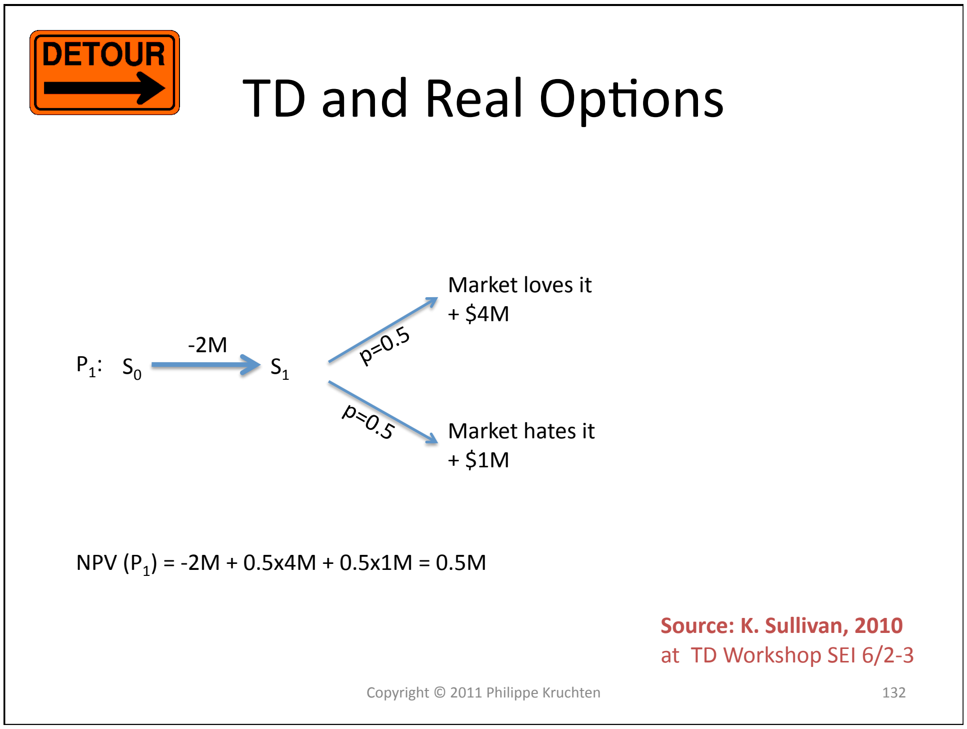
- In presence of technical debt:
  Cost of adding new features is higher

- When repaying (fixing), additional cost for retrofitting already implemented features

- Technical debt not repaid => lead to increased cost, forever
- Cost of fixing increases over time

M. Fowler

Copyright © 2011 Philippe Kruchten 131



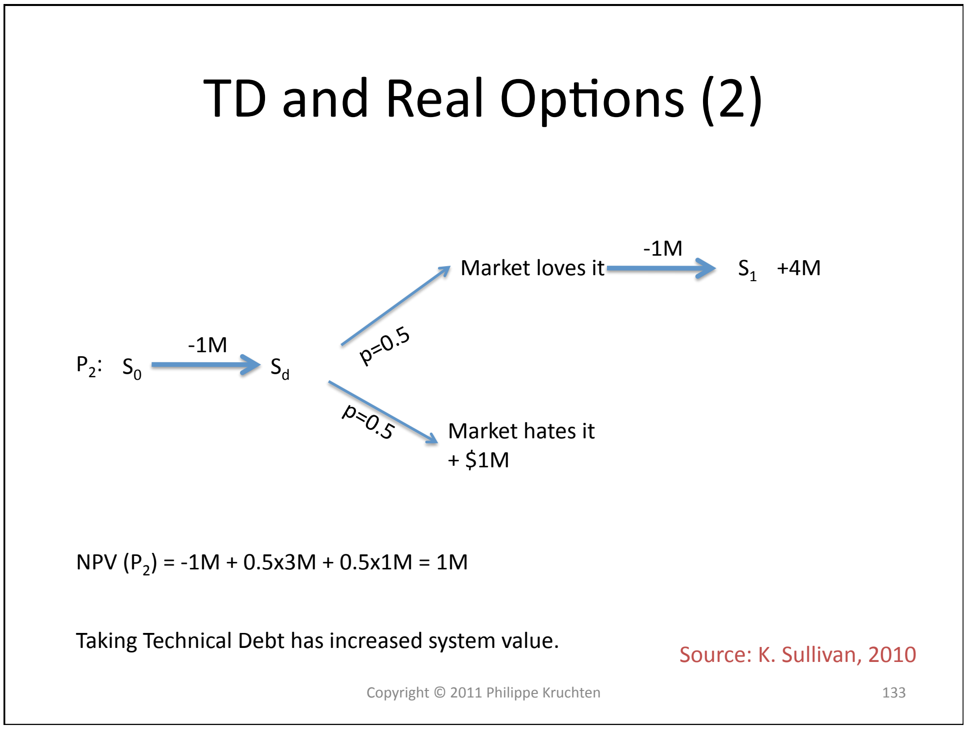
DETOUR

TD and Real Options

NPV (P_1) = -2M + 0.5x4M + 0.5x1M = 0.5M

Source: K. Sullivan, 2010
at TD Workshop SEI 6/2-3

132



TD and Real Options (2)

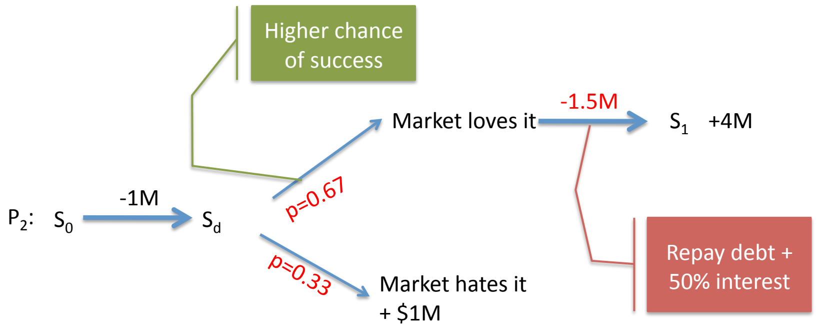
NPV (P_2) = -1M + 0.5x3M + 0.5x1M = 1M

Taking Technical Debt has increased system value.

Source: K. Sullivan, 2010

133

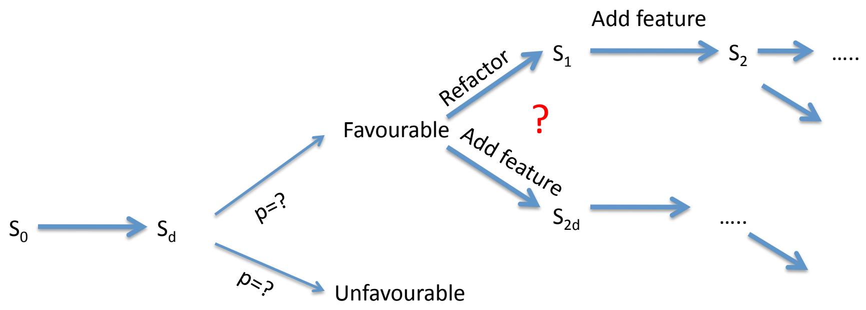
TD and Real Options (3)



NPV (P_3) = -1M + 0.67 x 2.5M + 0.33 x 1M = 1M

More realistically:
Debt + interest
High chances of success

TD and Real Options (4)



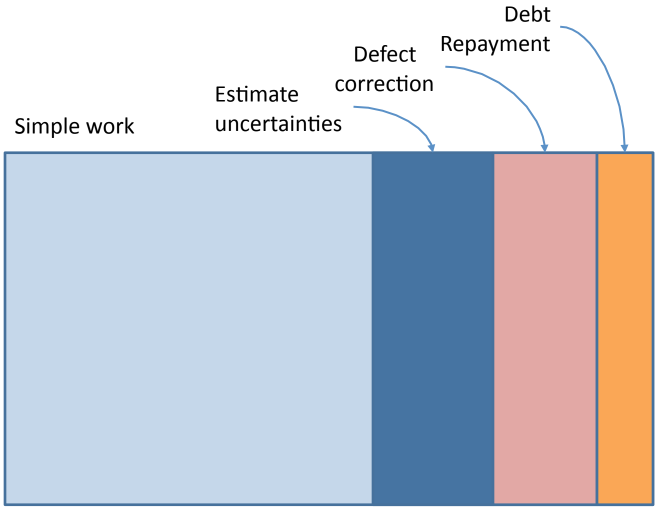
Not debt really, but options with different values…
Do we want to invest in architecture, in test, etc…

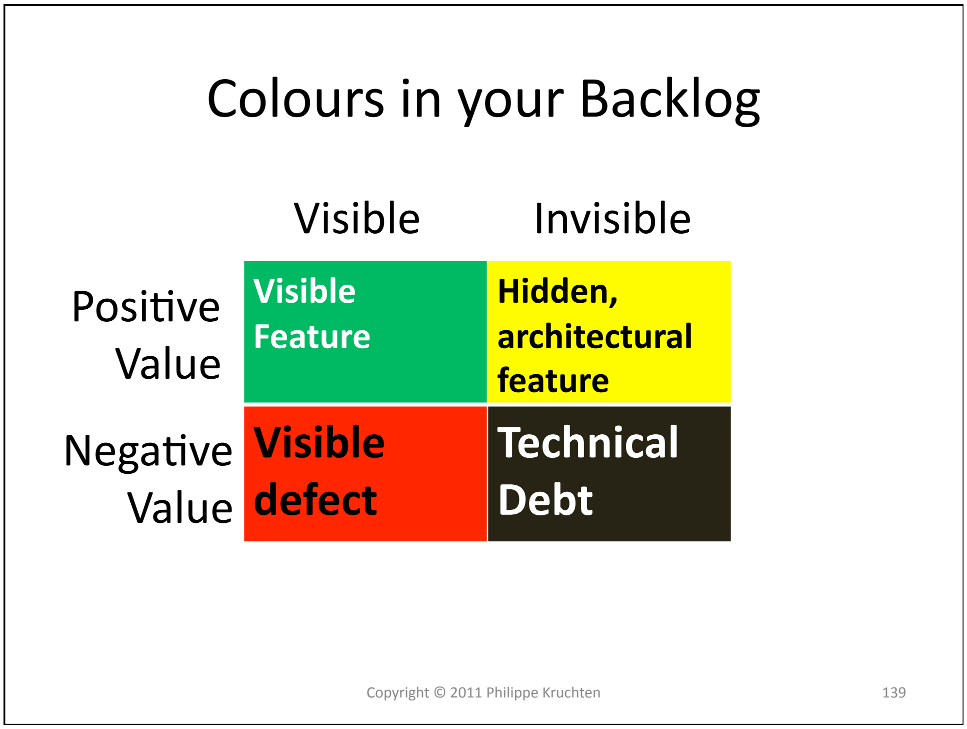
Source: K. Sullivan, 2010

- Technical debt is more a rhetorical category than a technical or ontological category.
  - The concept resonates well with the development community and the business community
  - Both sides “get” the metaphor.
- Technical debt is a concept that bridges the gap between:
  - Business decisions makers
  - Software designers/developers

DETOUR

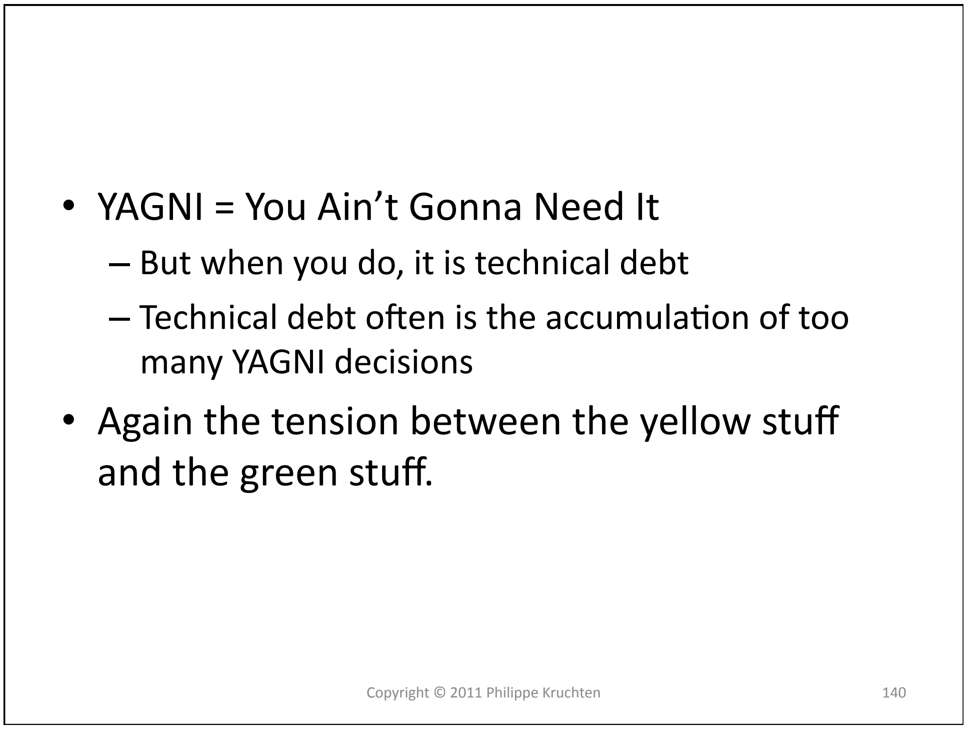
Buffer for debt repayment



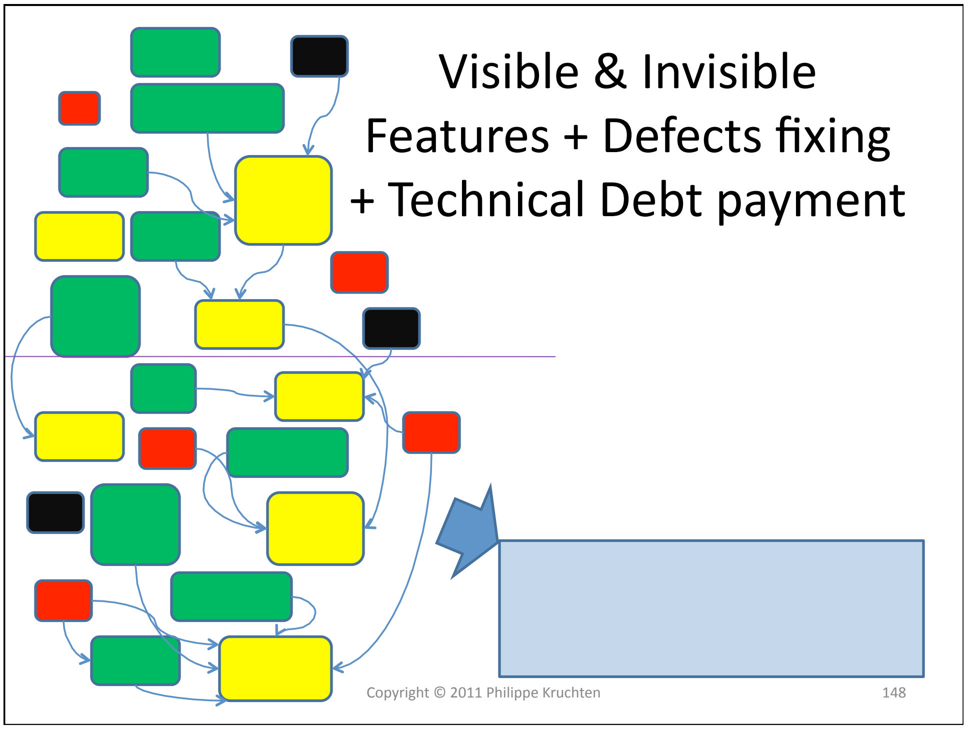


Colours in your Backlog

	Visible	Invisible
Positive Value	Visible Feature	Hidden, architectural feature
Negative Value	Visible defect	Technical Debt

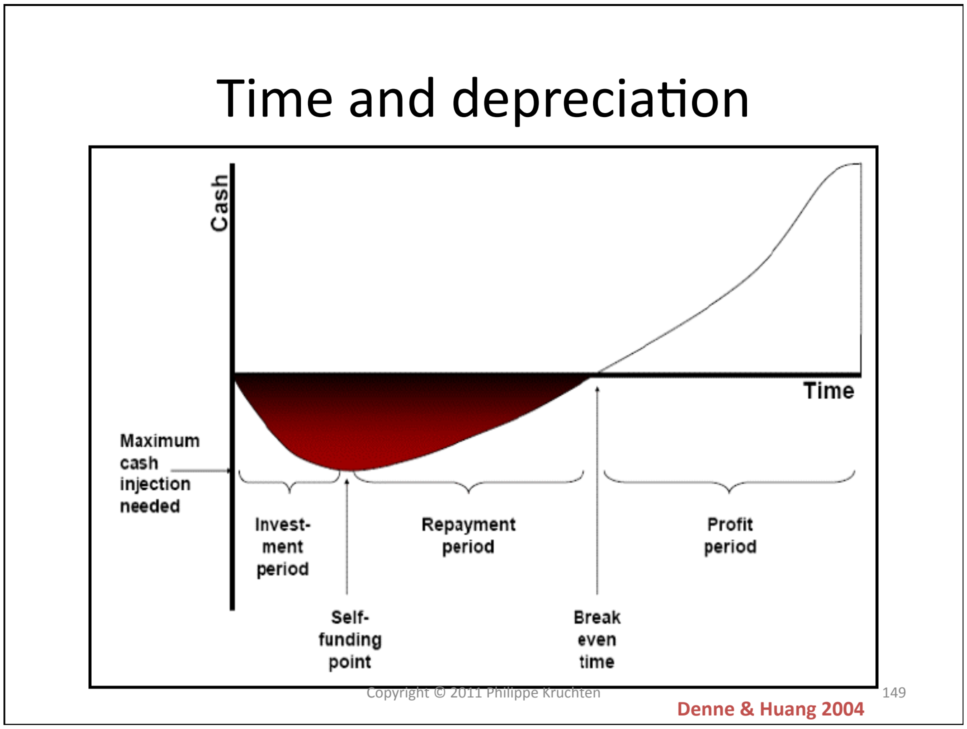


- YAGNI = You Ain’t Gonna Need It
  - But when you do, it is technical debt
  - Technical debt often is the accumulation of too many YAGNI decisions
- Again the tension between the yellow stuff and the green stuff.



Visible & Invisible Features + Defects fixing + Technical Debt payment

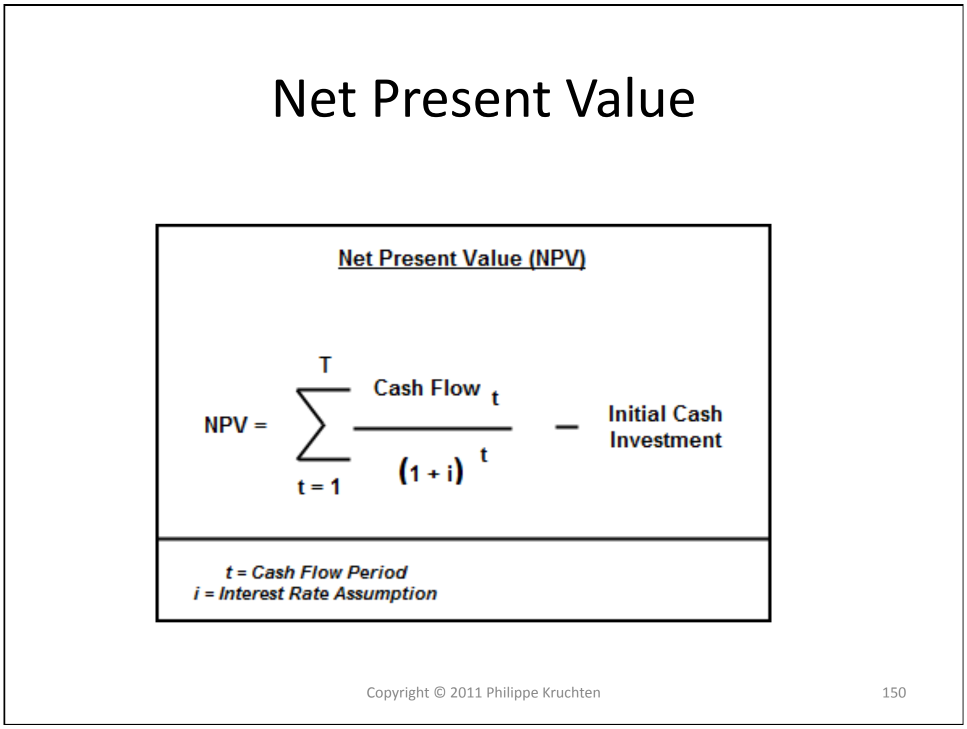
148



Time and depreciation

149

Denne & Huang 2004



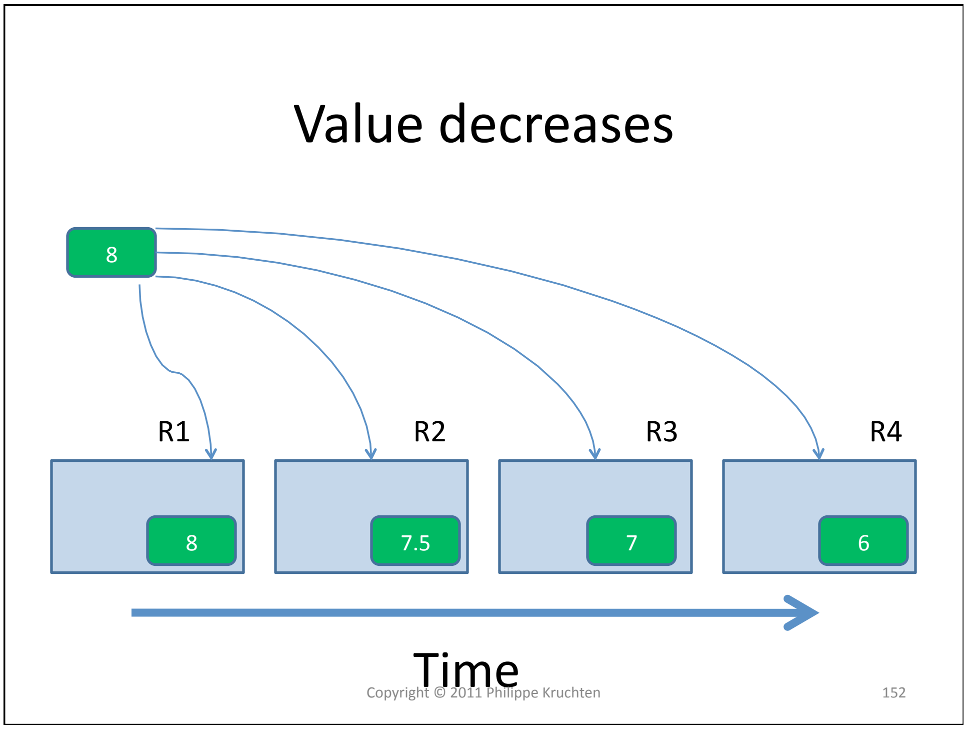
Net Present Value

Net Present Value (NPV)

$$\text{NPV} = \sum_{t=1}^{T} \frac{\text{Cash Flow}_t}{(1+i)^t} - \text{Initial Cash Investment}$$

t = Cash Flow Period
i = Interest Rate Assumption

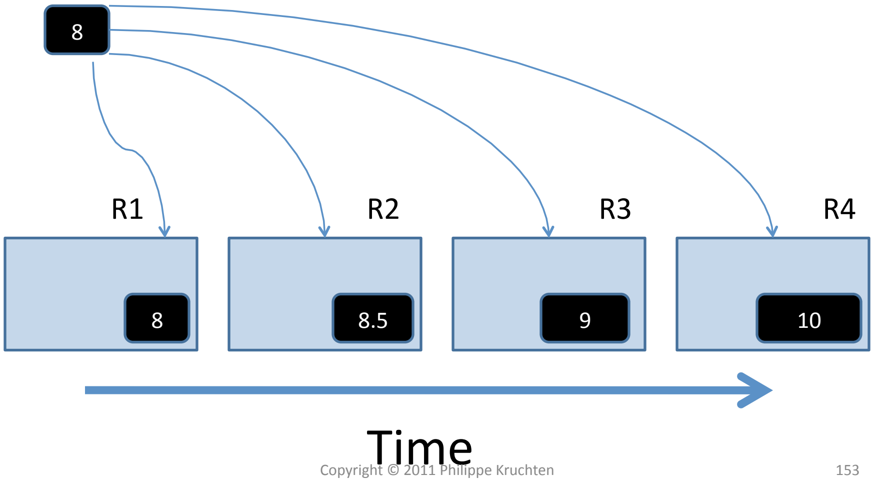
150

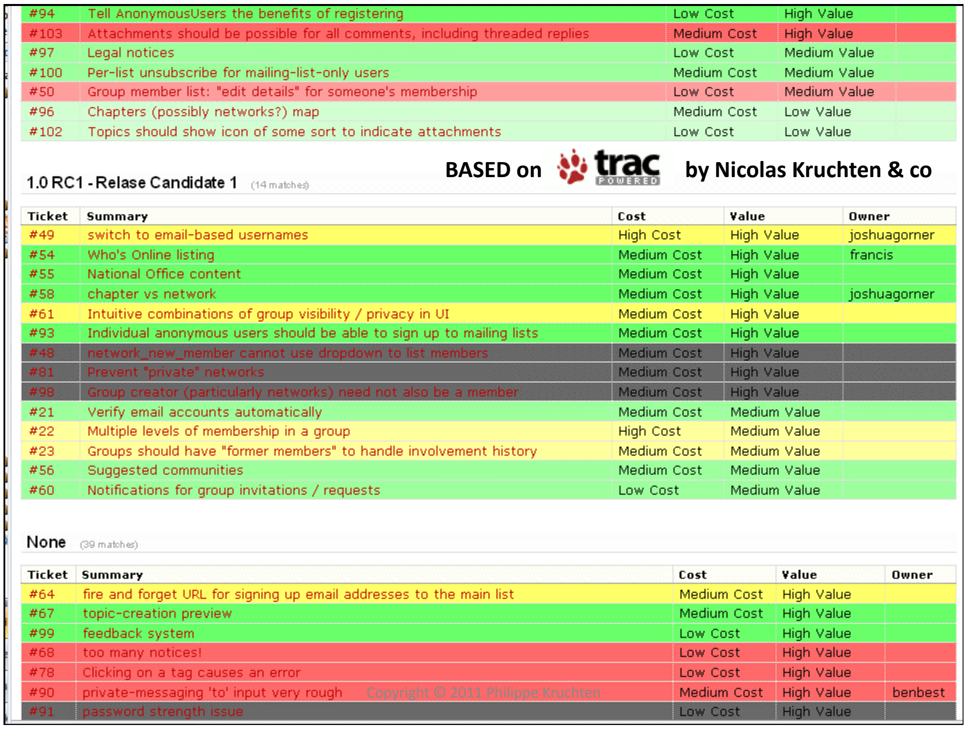


Value decreases

152

Technical debt: increase (?)





#94	Tell AnonymousUsers the benefits of registering	Low Cost	High Value	
#103	Attachments should be possible for all comments, including threaded replies	Medium Cost	High Value	
#97	Legal notices	Low Cost	Medium Value	
#100	Per-list unsubscribe for mailing-list-only users	Medium Cost	Medium Value	
#50	Group member list: "edit details" for someone's membership	Low Cost	Medium Value	
#96	Chapters (possibly networks?) map	Medium Cost	Low Value	
#102	Topics should show icon of some sort to indicate attachments	Low Cost	Low Value	

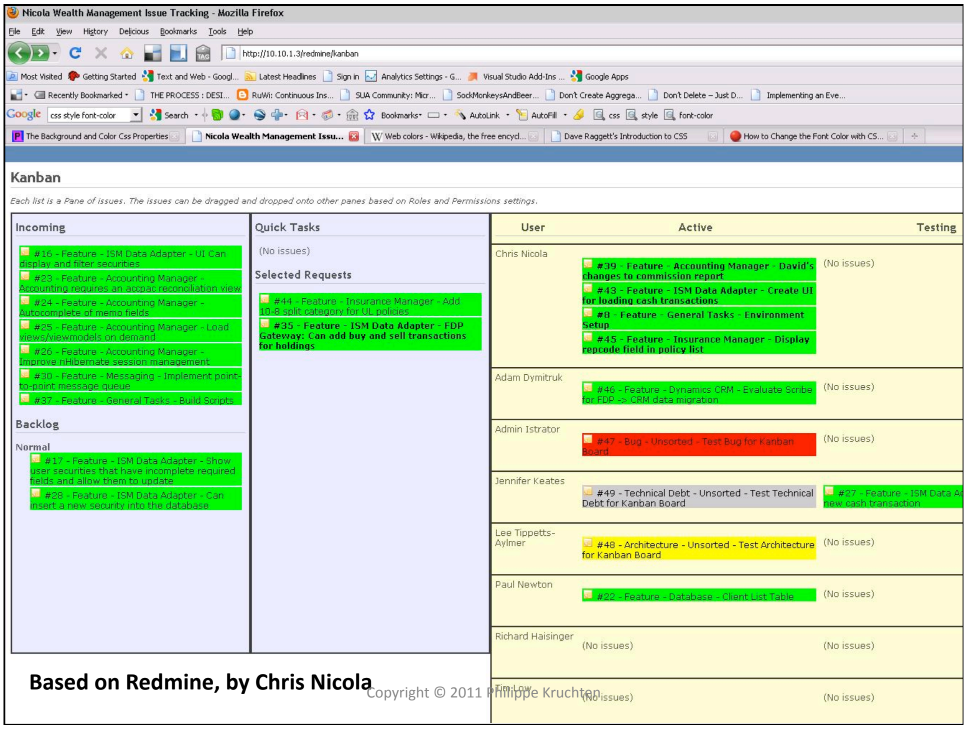
BASED on trac POWERED **by Nicolas Kruchten & co**

1.0 RC1 - Relase Candidate 1 (14 matches)

Ticket	Summary	Cost	Value	Owner
#49	switch to email-based usernames	High Cost	High Value	joshuagorner
#54	Who's Online listing	Medium Cost	High Value	francis
#55	National Office content	Medium Cost	High Value	
#58	chapter vs network	Medium Cost	High Value	joshuagorner
#61	Intuitive combinations of group visibility / privacy in UI	Medium Cost	High Value	
#93	Individual anonymous users should be able to sign up to mailing lists	Medium Cost	High Value	
#48	network_new_member cannot use dropdown to list members	Medium Cost	High Value	
#81	Prevent "private" networks	Medium Cost	High Value	
#98	Group creator (particularly networks) need not also be a member	Medium Cost	High Value	
#21	Verify email accounts automatically	Medium Cost	Medium Value	
#22	Multiple levels of membership in a group	High Cost	Medium Value	
#23	Groups should have "former members" to handle involvement history	Medium Cost	Medium Value	
#56	Suggested communities	Medium Cost	Medium Value	
#60	Notifications for group invitations / requests	Low Cost	Medium Value	

None (39 matches)

Ticket	Summary	Cost	Value	Owner
#64	fire and forget URL for signing up email addresses to the main list	Medium Cost	High Value	
#67	topic-creation preview	Medium Cost	High Value	
#99	feedback system	Low Cost	High Value	
#68	too many notices!	Low Cost	High Value	
#78	Clicking on a tag causes an error	Low Cost	High Value	
#90	private-messaging 'to' input very rough	Medium Cost	High Value	benbest
#91	password strength issue	Low Cost	High Value	



Nicola Wealth Management Issue Tracking - Mozilla Firefox
File Edit View History Delicious Bookmarks Tools Help
http://10.10.1.3/redmine/kanban
Most Visited Getting Started Text and Web - Googl... Latest Headlines Sign in Analytics Settings - G... Visual Studio Add-Ins ... Google Apps
Recently Bookmarked THE PROCESS : DESI... RuWi: Continuous Ins... SUA Community: Micr... SockMonkeysAndBeer... Don't Create Aggrega... Don't Delete – Just D... Implementing an Eve...
Google css style font-color Search Bookmarks AutoLink AutoFill css style font-color
The Background and Color Css Properties Nicola Wealth Management Issu... Web colors - Wikipedia, the free encycl... Dave Raggett's Introduction to CSS How to Change the Font Color with CS...

Kanban

Each list is a Pane of issues. The issues can be dragged and dropped onto other panes based on Roles and Permissions settings.

Incoming

- #16 - Feature - ISM Data Adapter - UI Can display and filter securities
- #23 - Feature - Accounting Manager - Accounting requires an accpac reconciliation view
- #24 - Feature - Accounting Manager - Autocomplete of memo fields
- #25 - Feature - Accounting Manager - Load views/viewmodels on demand
- #26 - Feature - Accounting Manager - Improve nHibernate session management
- #30 - Feature - Messaging - Implement point-to-point message queue
- #37 - Feature - General Tasks - Build Scripts

Backlog

Normal

- #17 - Feature - ISM Data Adapter - Show user securities that have incomplete required fields and allow them to update
- #28 - Feature - ISM Data Adapter - Can insert a new security into the database

Quick Tasks

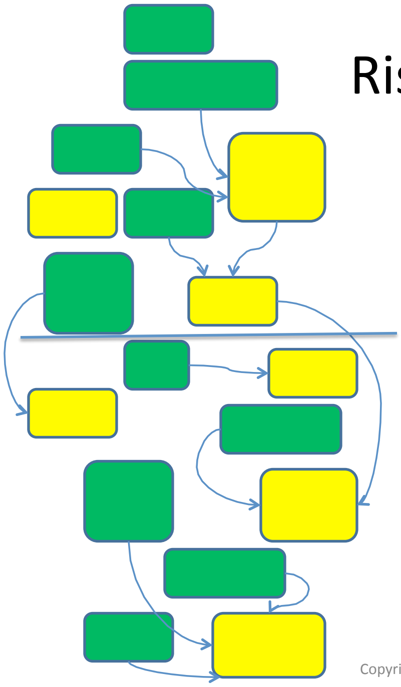
(No issues)

Selected Requests

- #44 - Feature - Insurance Manager - Add 10-8 split category for UL policies
- **#35 - Feature - ISM Data Adapter - FDP Gateway: Can add buy and sell transactions for holdings**

User	Active	Testing
Chris Nicola	**#39 - Feature - Accounting Manager - David's changes to commission report** **#43 - Feature - ISM Data Adapter - Create UI for loading cash transactions** **#8 - Feature - General Tasks - Environment Setup** **#45 - Feature - Insurance Manager - Display repcode field in policy list**	(No issues)
Adam Dymitruk	#46 - Feature - Dynamics CRM - Evaluate Scribe for FDP -> CRM data migration	(No issues)
Admin Istrator	#47 - Bug - Unsorted - Test Bug for Kanban Board	(No issues)
Jennifer Keates	#49 - Technical Debt - Unsorted - Test Technical Debt for Kanban Board	#27 - Feature - ISM Data Ad new cash transaction
Lee Tippetts-Aylmer	#48 - Architecture - Unsorted - Test Architecture for Kanban Board	(No issues)
Paul Newton	#22 - Feature - Database - Client List Table	(No issues)
Richard Haisinger	(No issues)	(No issues)
Tim Low	(No issues)	(No issues)

Based on Redmine, by Chris Nicola

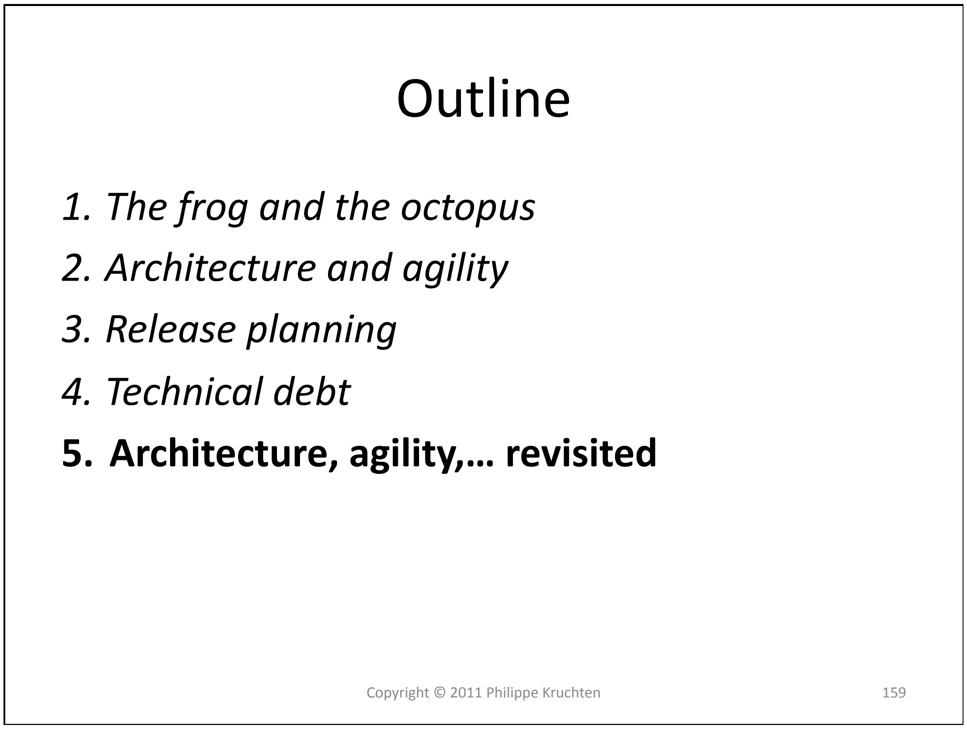


Risks & Uncertainties

Rule of thumb:

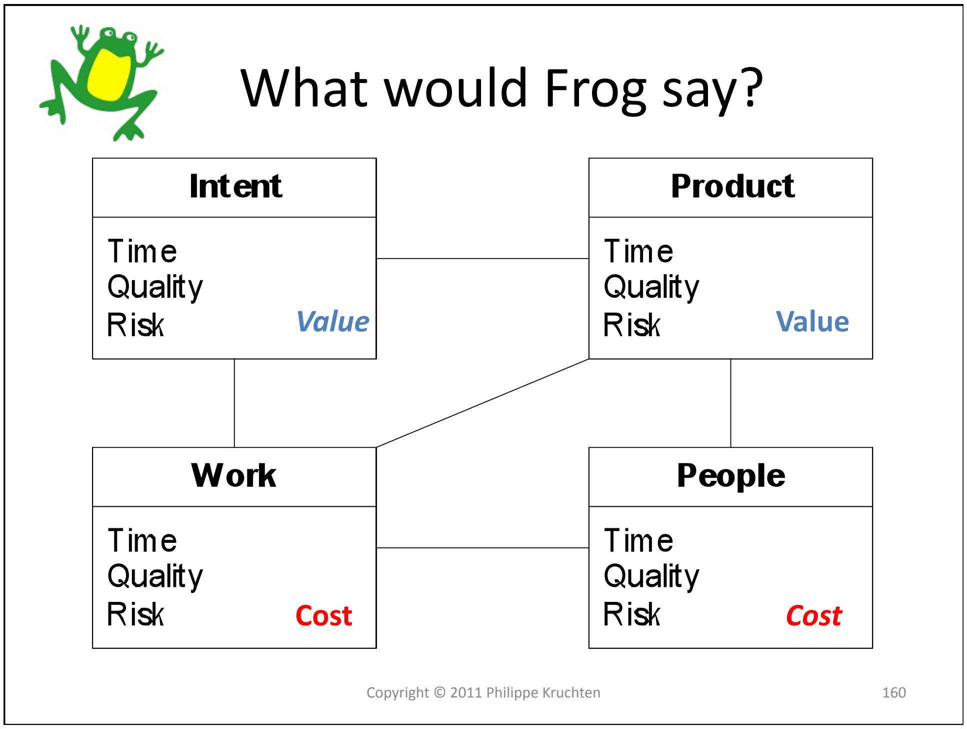
- Green stuff: move up
  - defer
- Yellow stuff: move down
  - Experiment now

Karl Wiegers, 1999
RUP, 1998



Outline

1. *The frog and the octopus*
2. *Architecture and agility*
3. *Release planning*
4. *Technical debt*
5. **Architecture, agility,… revisited**



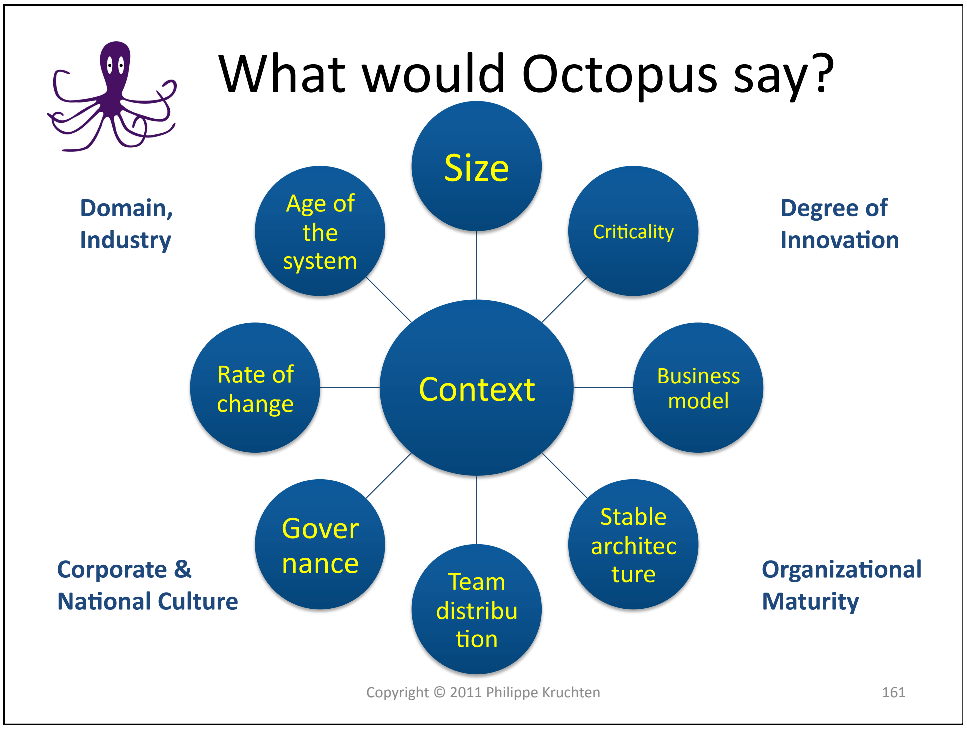
What would Frog say?

Intent
Time
Quality
Risk *Value*

Product
Time
Quality
Risk **Value**

Work
Time
Quality
Risk **Cost**

People
Time
Quality
Risk ***Cost***



What would Octopus say?

Context

Size

Criticality

Business model

Stable architecture

Team distribution

Governance

Rate of change

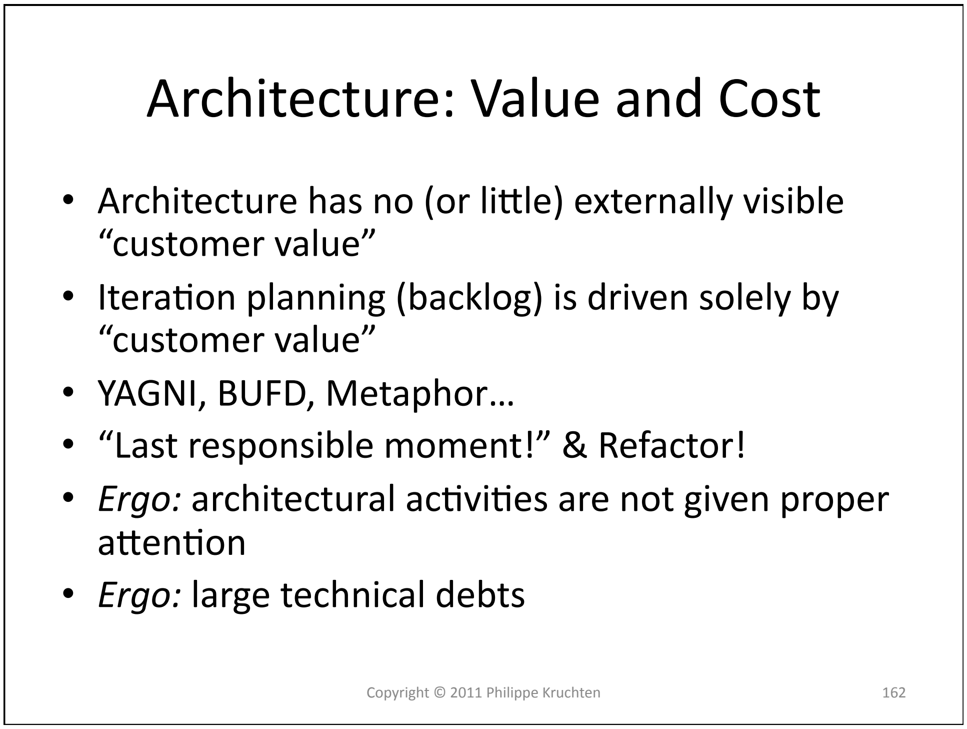
Age of the system

Domain, Industry

Degree of Innovation

Organizational Maturity

Corporate & National Culture



Architecture: Value and Cost

- Architecture has no (or little) externally visible "customer value"
- Iteration planning (backlog) is driven solely by "customer value"
- YAGNI, BUFD, Metaphor…
- "Last responsible moment!" & Refactor!
- *Ergo:* architectural activities are not given proper attention
- *Ergo:* large technical debts

Role of Architecture

- Novel system
- Gradual emergence of architecture
- Validation of architecture with actual functionality
- Early enough to support development

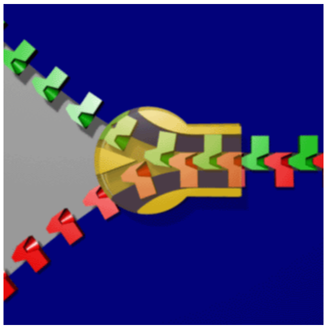
Zipper model…

- Not just BUFD
- No YAGNI effect

163

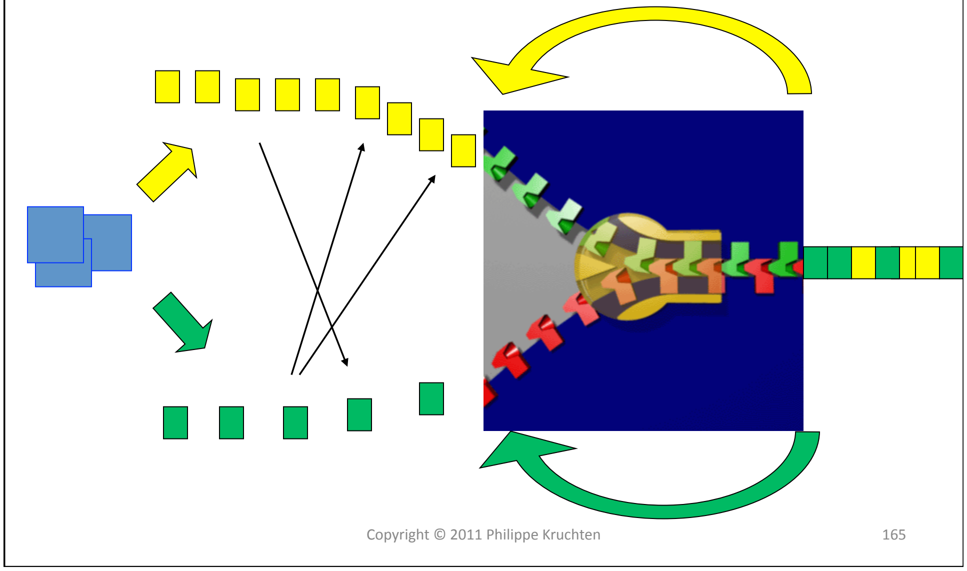
Planning

- From requirements derive:
  - Architectural requirements
  - Functional requirements
- Establish
  - Dependencies
  - Cost
- Plan interleaving:
  - Functional increments
  - Architectural increments



164

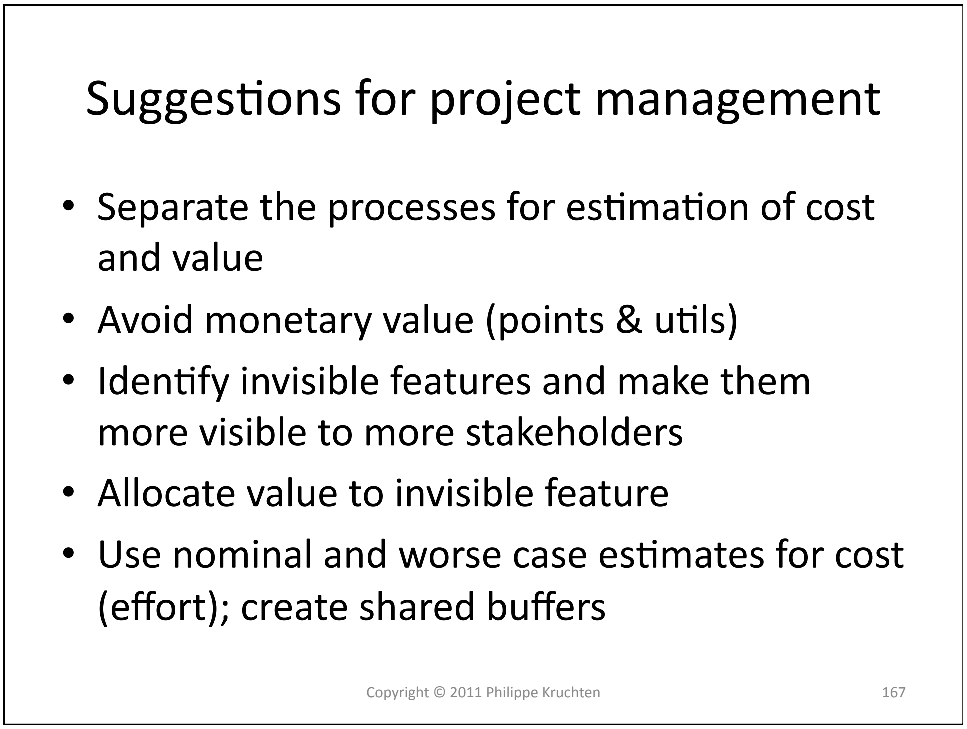
Zipper: Weaving the functional and architectural work items



Benefits

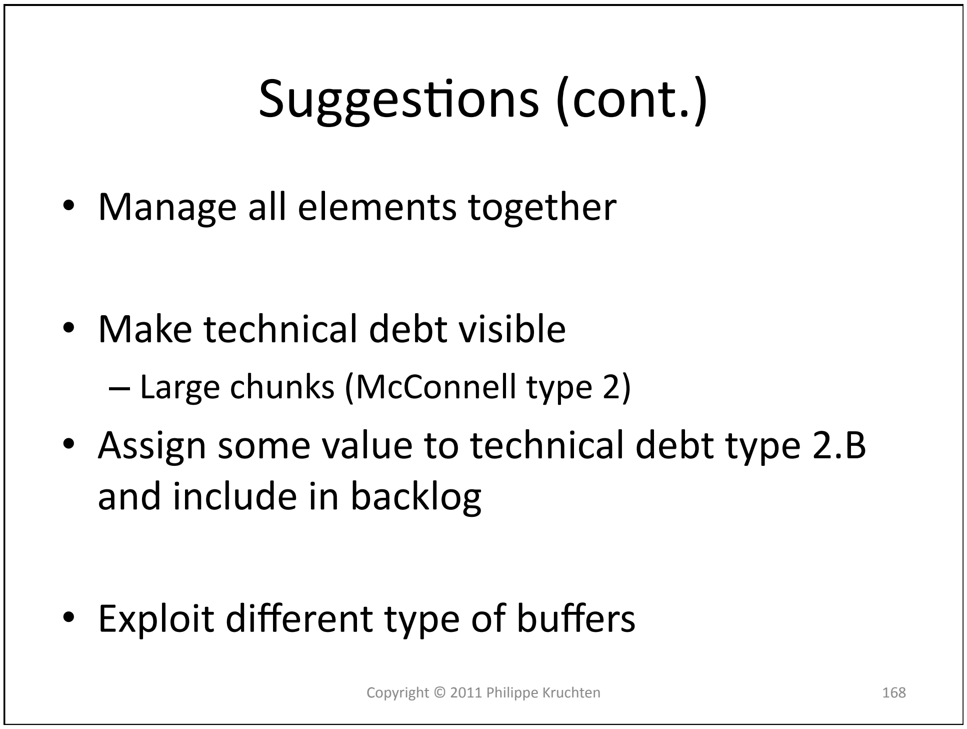
- Gradual emergence of architecture
- Validation of architecture with actual functionality
- Early enough to support development

- Not just BUFD
- No YAGNI effect



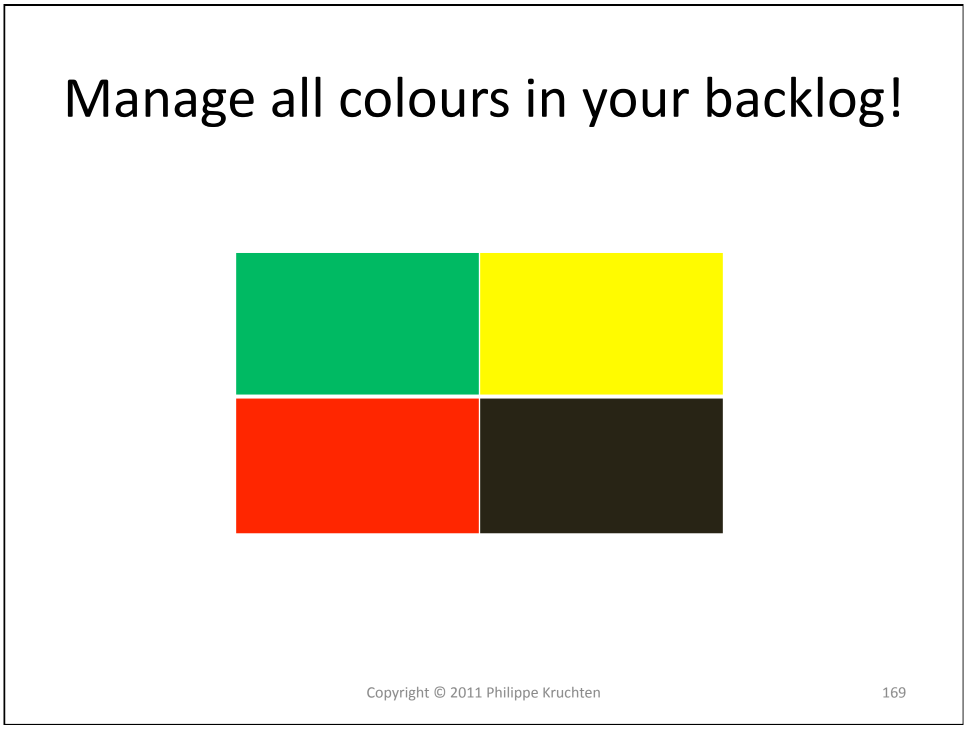
Suggestions for project management

- Separate the processes for estimation of cost and value
- Avoid monetary value (points & utils)
- Identify invisible features and make them more visible to more stakeholders
- Allocate value to invisible feature
- Use nominal and worse case estimates for cost (effort); create shared buffers



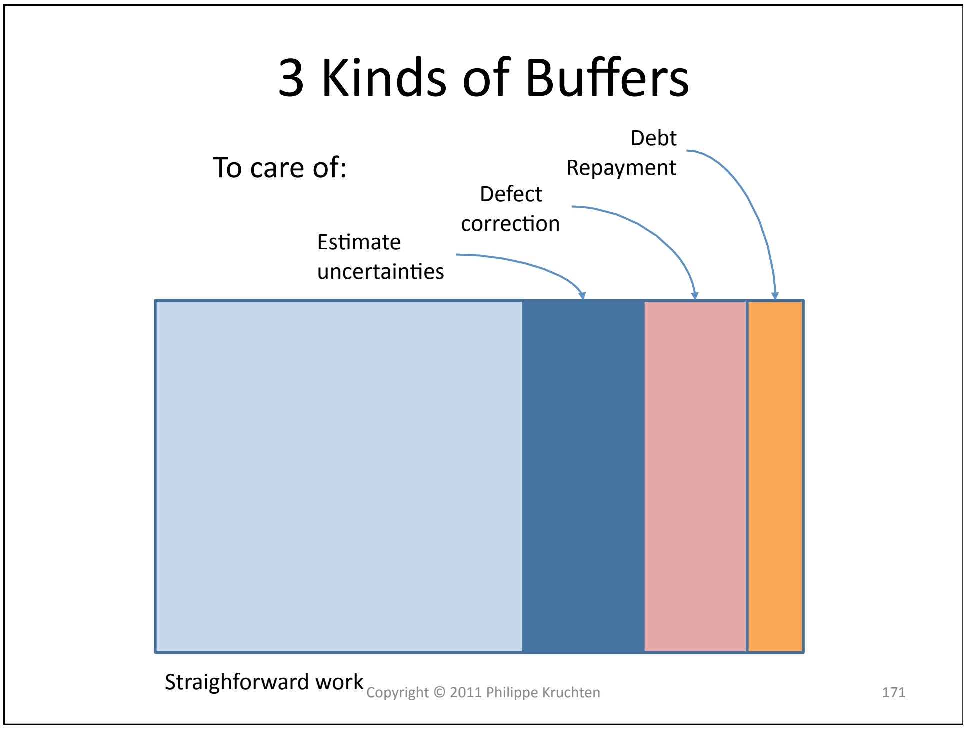
Suggestions (cont.)

- Manage all elements together
- Make technical debt visible
  - Large chunks (McConnell type 2)
- Assign some value to technical debt type 2.B and include in backlog
- Exploit different type of buffers



Manage all colours in your backlog!

Copyright © 2011 Philippe Kruchten 169



3 Kinds of Buffers

To care of:

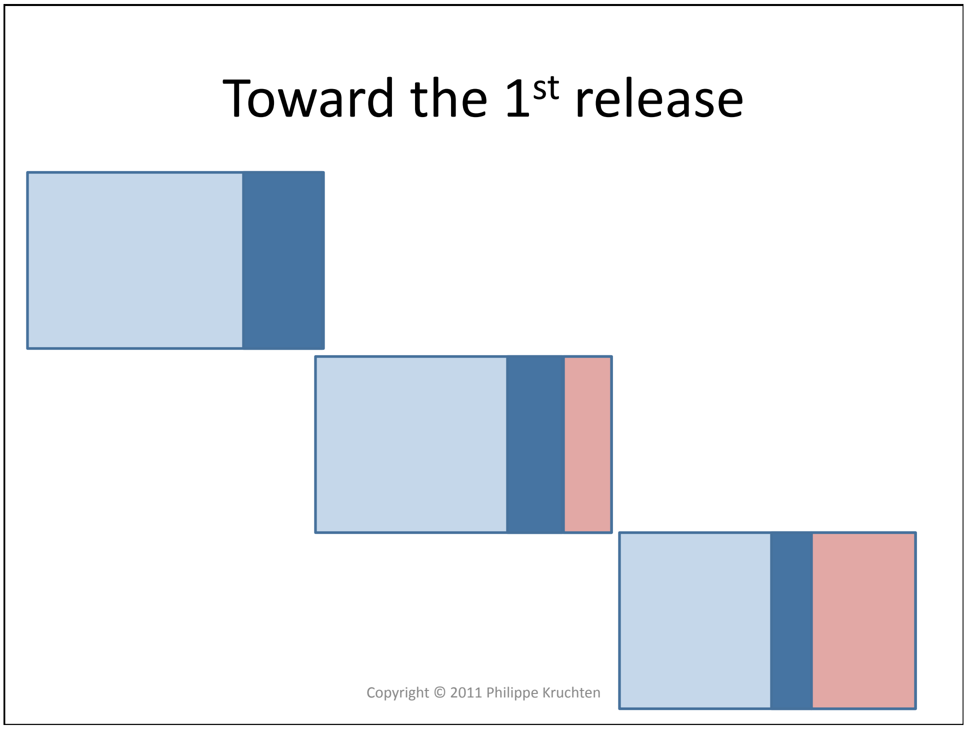
Estimate uncertainties

Defect correction

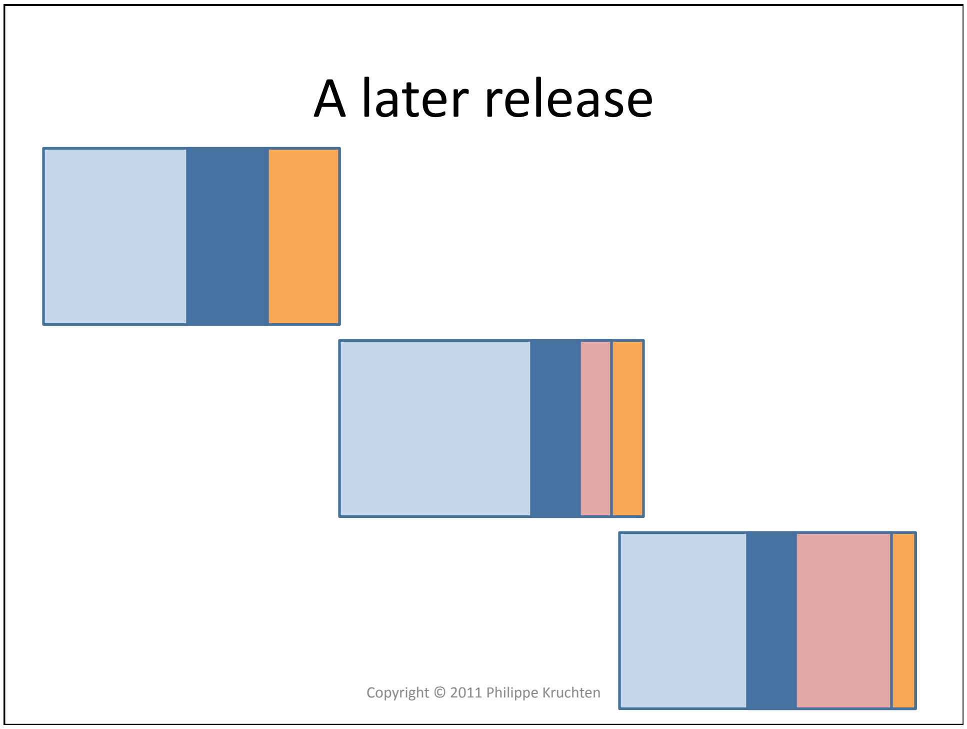
Debt Repayment

Straighforward work

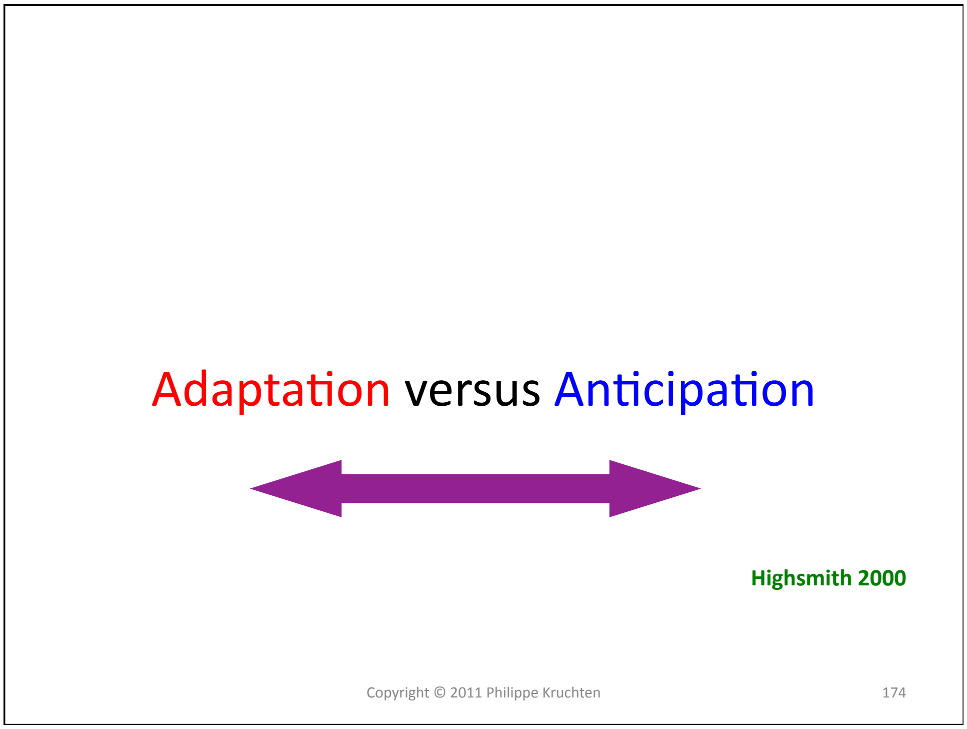
Copyright © 2011 Philippe Kruchten 171



Toward the 1st release

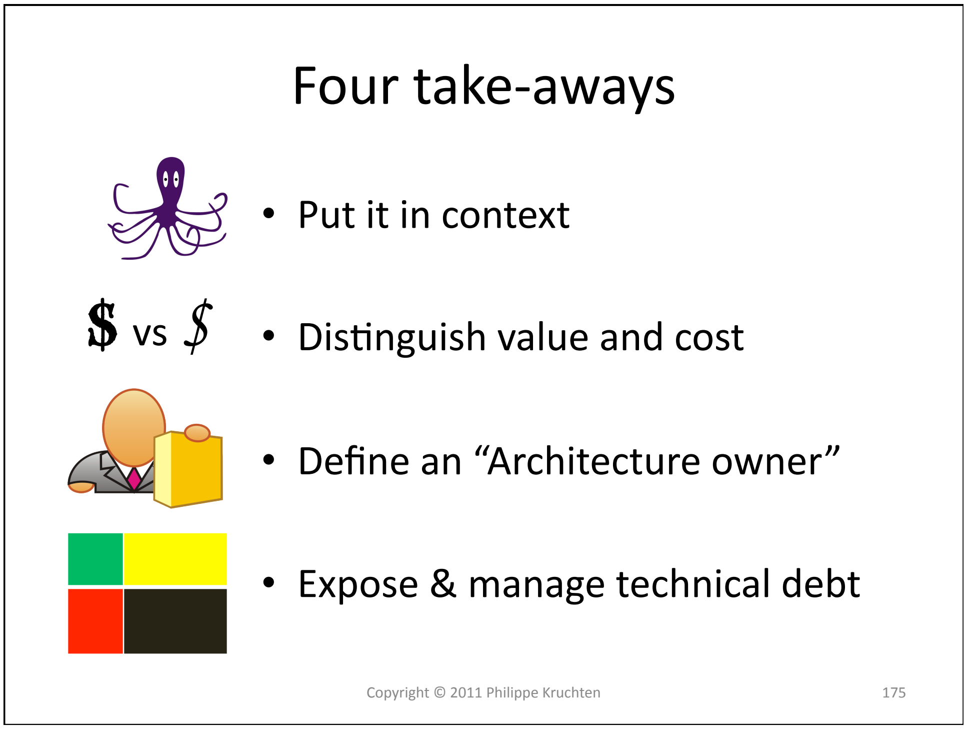


A later release



Adaptation versus Anticipation

Highsmith 2000



Four take-aways

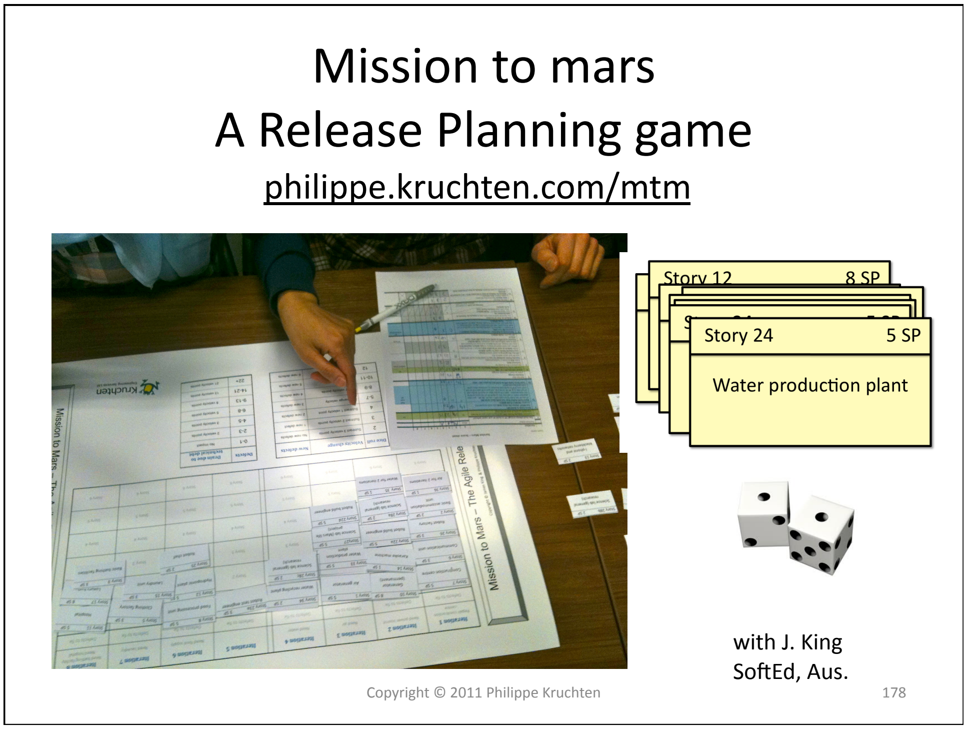
$ vs $

- Put it in context
- Distinguish value and cost
- Define an "Architecture owner"
- Expose & manage technical debt

Game to Introduce Tech Debt

www.sei.cmu.edu/architecture/tools/hardchoices/

177

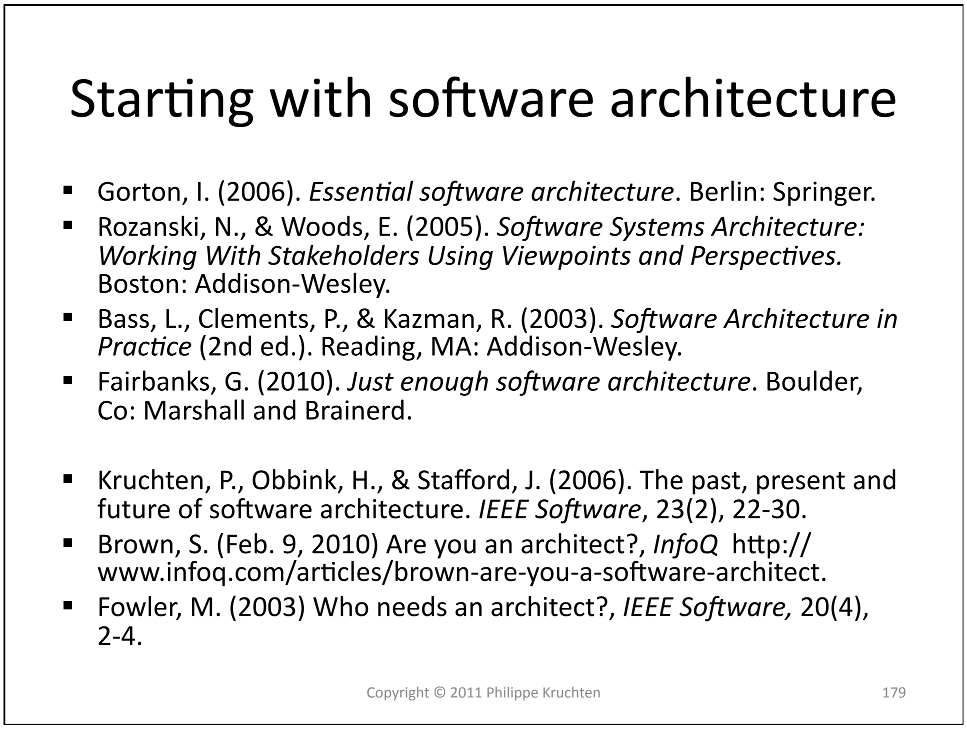


Mission to mars
A Release Planning game

philippe.kruchten.com/mtm

with J. King
SoftEd, Aus.

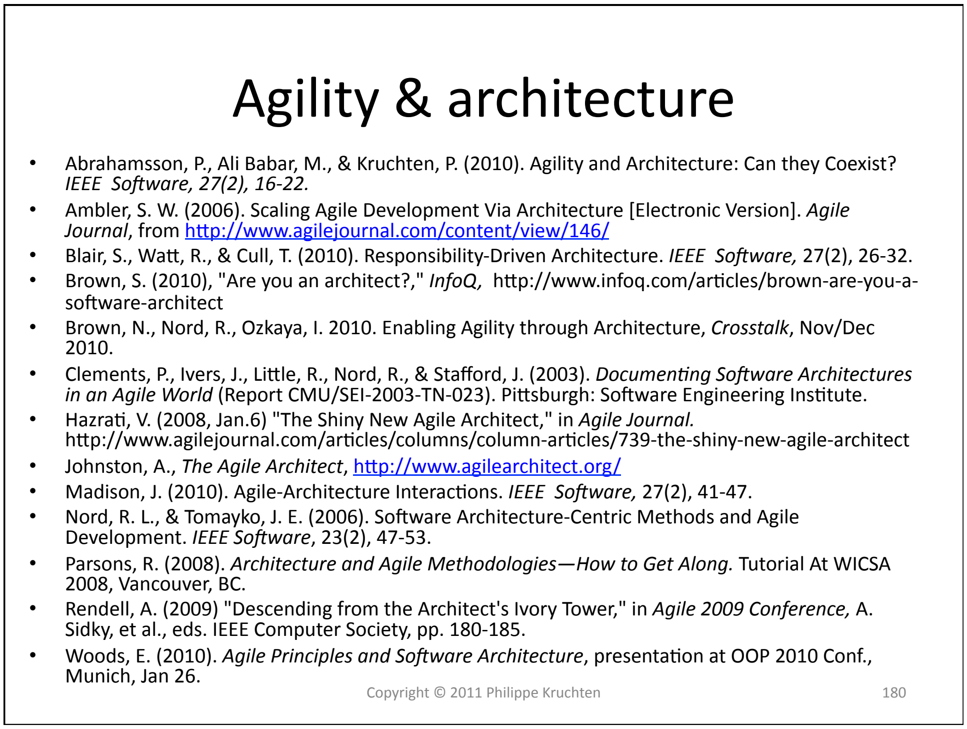
178



Starting with software architecture

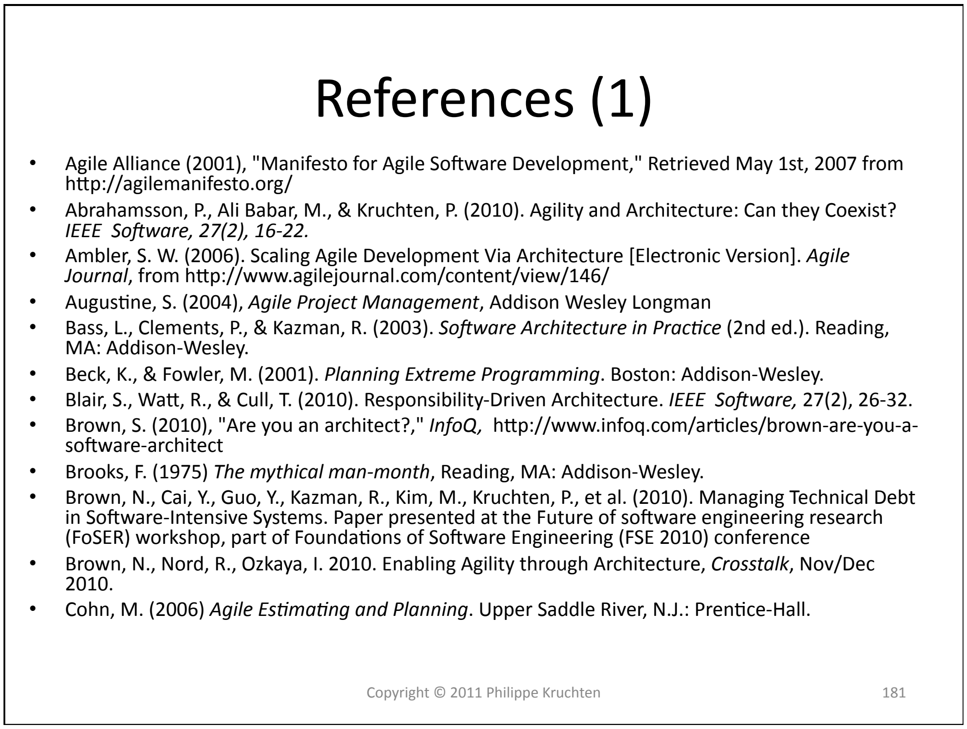
- Gorton, I. (2006). *Essential software architecture*. Berlin: Springer.
- Rozanski, N., & Woods, E. (2005). *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.* Boston: Addison-Wesley.
- Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice* (2nd ed.). Reading, MA: Addison-Wesley.
- Fairbanks, G. (2010). *Just enough software architecture*. Boulder, Co: Marshall and Brainerd.

- Kruchten, P., Obbink, H., & Stafford, J. (2006). The past, present and future of software architecture. *IEEE Software*, 23(2), 22-30.
- Brown, S. (Feb. 9, 2010) Are you an architect?, *InfoQ* http://www.infoq.com/articles/brown-are-you-a-software-architect.
- Fowler, M. (2003) Who needs an architect?, *IEEE Software,* 20(4), 2-4.



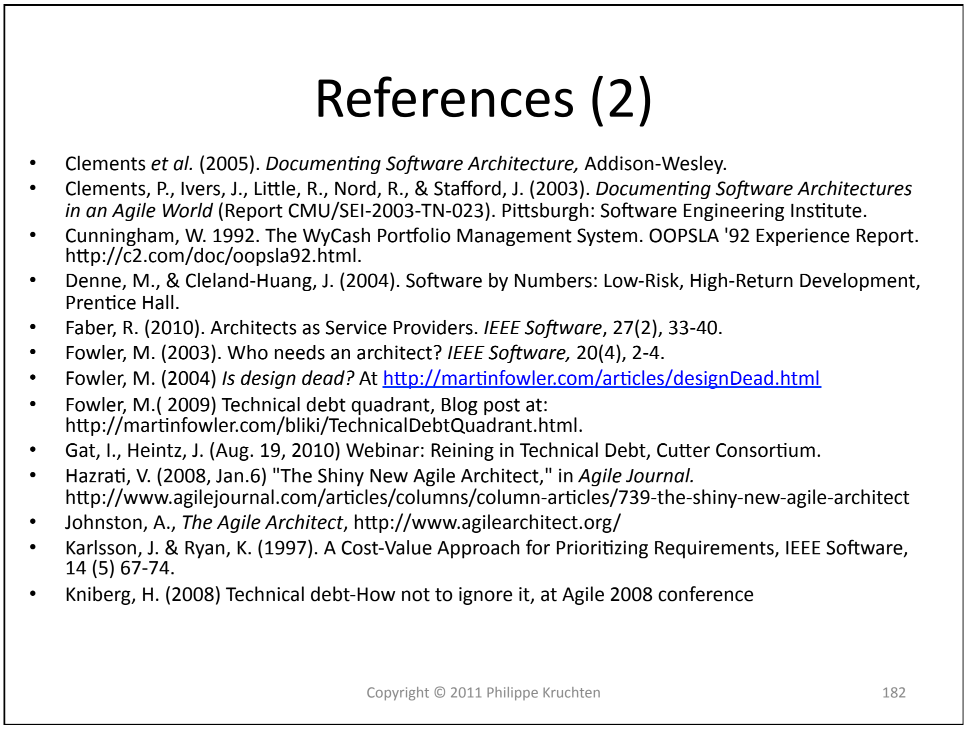
Agility & architecture

- Abrahamsson, P., Ali Babar, M., & Kruchten, P. (2010). Agility and Architecture: Can they Coexist? *IEEE Software, 27(2), 16-22.*
- Ambler, S. W. (2006). Scaling Agile Development Via Architecture [Electronic Version]. *Agile Journal*, from http://www.agilejournal.com/content/view/146/
- Blair, S., Watt, R., & Cull, T. (2010). Responsibility-Driven Architecture. *IEEE Software,* 27(2), 26-32.
- Brown, S. (2010), "Are you an architect?," *InfoQ,* http://www.infoq.com/articles/brown-are-you-a-software-architect
- Brown, N., Nord, R., Ozkaya, I. 2010. Enabling Agility through Architecture, *Crosstalk*, Nov/Dec 2010.
- Clements, P., Ivers, J., Little, R., Nord, R., & Stafford, J. (2003). *Documenting Software Architectures in an Agile World* (Report CMU/SEI-2003-TN-023). Pittsburgh: Software Engineering Institute.
- Hazrati, V. (2008, Jan.6) "The Shiny New Agile Architect," in *Agile Journal.* http://www.agilejournal.com/articles/columns/column-articles/739-the-shiny-new-agile-architect
- Johnston, A., *The Agile Architect*, http://www.agilearchitect.org/
- Madison, J. (2010). Agile-Architecture Interactions. *IEEE Software,* 27(2), 41-47.
- Nord, R. L., & Tomayko, J. E. (2006). Software Architecture-Centric Methods and Agile Development. *IEEE Software*, 23(2), 47-53.
- Parsons, R. (2008). *Architecture and Agile Methodologies—How to Get Along.* Tutorial At WICSA 2008, Vancouver, BC.
- Rendell, A. (2009) "Descending from the Architect's Ivory Tower," in *Agile 2009 Conference,* A. Sidky, et al., eds. IEEE Computer Society, pp. 180-185.
- Woods, E. (2010). *Agile Principles and Software Architecture*, presentation at OOP 2010 Conf., Munich, Jan 26.



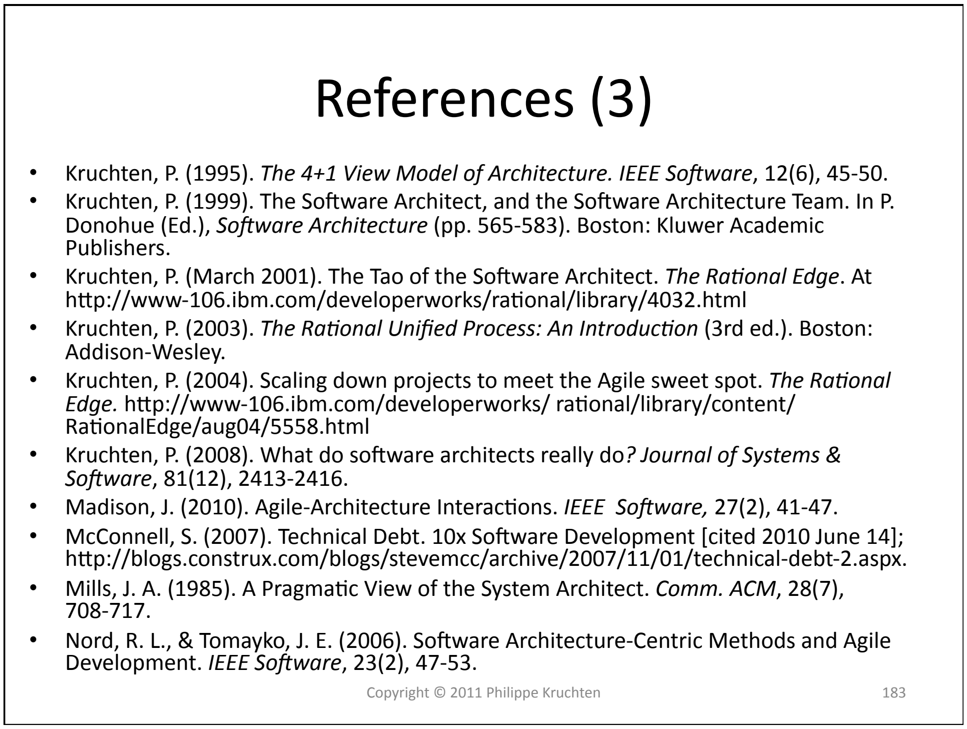
References (1)

- Agile Alliance (2001), "Manifesto for Agile Software Development," Retrieved May 1st, 2007 from http://agilemanifesto.org/
- Abrahamsson, P., Ali Babar, M., & Kruchten, P. (2010). Agility and Architecture: Can they Coexist? *IEEE Software, 27(2), 16-22.*
- Ambler, S. W. (2006). Scaling Agile Development Via Architecture [Electronic Version]. *Agile Journal*, from http://www.agilejournal.com/content/view/146/
- Augustine, S. (2004), *Agile Project Management*, Addison Wesley Longman
- Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice* (2nd ed.). Reading, MA: Addison-Wesley.
- Beck, K., & Fowler, M. (2001). *Planning Extreme Programming*. Boston: Addison-Wesley.
- Blair, S., Watt, R., & Cull, T. (2010). Responsibility-Driven Architecture. *IEEE Software,* 27(2), 26-32.
- Brown, S. (2010), "Are you an architect?," *InfoQ,* http://www.infoq.com/articles/brown-are-you-a-software-architect
- Brooks, F. (1975) *The mythical man-month*, Reading, MA: Addison-Wesley.
- Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., et al. (2010). Managing Technical Debt in Software-Intensive Systems. Paper presented at the Future of software engineering research (FoSER) workshop, part of Foundations of Software Engineering (FSE 2010) conference
- Brown, N., Nord, R., Ozkaya, I. 2010. Enabling Agility through Architecture, *Crosstalk*, Nov/Dec 2010.
- Cohn, M. (2006) *Agile Estimating and Planning*. Upper Saddle River, N.J.: Prentice-Hall.



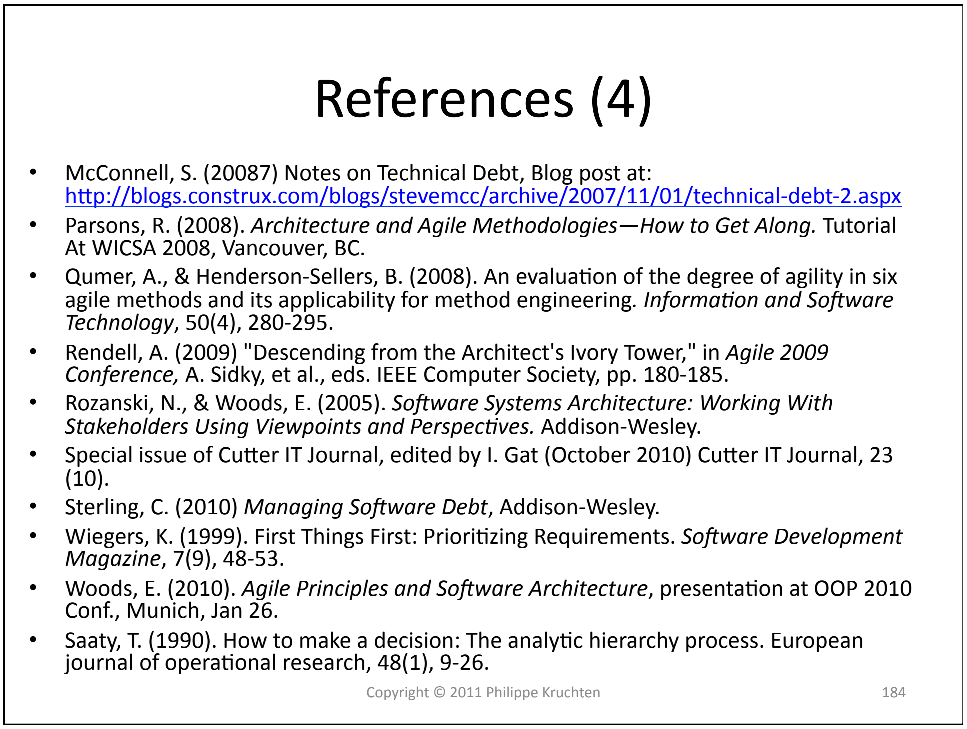
References (2)

- Clements *et al.* (2005). *Documenting Software Architecture,* Addison-Wesley.
- Clements, P., Ivers, J., Little, R., Nord, R., & Stafford, J. (2003). *Documenting Software Architectures in an Agile World* (Report CMU/SEI-2003-TN-023). Pittsburgh: Software Engineering Institute.
- Cunningham, W. 1992. The WyCash Portfolio Management System. OOPSLA '92 Experience Report. http://c2.com/doc/oopsla92.html.
- Denne, M., & Cleland-Huang, J. (2004). Software by Numbers: Low-Risk, High-Return Development, Prentice Hall.
- Faber, R. (2010). Architects as Service Providers. *IEEE Software*, 27(2), 33-40.
- Fowler, M. (2003). Who needs an architect? *IEEE Software,* 20(4), 2-4.
- Fowler, M. (2004) *Is design dead?* At http://martinfowler.com/articles/designDead.html
- Fowler, M.(2009) Technical debt quadrant, Blog post at: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html.
- Gat, I., Heintz, J. (Aug. 19, 2010) Webinar: Reining in Technical Debt, Cutter Consortium.
- Hazrati, V. (2008, Jan.6) "The Shiny New Agile Architect," in *Agile Journal.* http://www.agilejournal.com/articles/columns/column-articles/739-the-shiny-new-agile-architect
- Johnston, A., *The Agile Architect*, http://www.agilearchitect.org/
- Karlsson, J. & Ryan, K. (1997). A Cost-Value Approach for Prioritizing Requirements, IEEE Software, 14 (5) 67-74.
- Kniberg, H. (2008) Technical debt-How not to ignore it, at Agile 2008 conference



References (3)

- Kruchten, P. (1995). *The 4+1 View Model of Architecture. IEEE Software*, 12(6), 45-50.
- Kruchten, P. (1999). The Software Architect, and the Software Architecture Team. In P. Donohue (Ed.), *Software Architecture* (pp. 565-583). Boston: Kluwer Academic Publishers.
- Kruchten, P. (March 2001). The Tao of the Software Architect. *The Rational Edge*. At http://www-106.ibm.com/developerworks/rational/library/4032.html
- Kruchten, P. (2003). *The Rational Unified Process: An Introduction* (3rd ed.). Boston: Addison-Wesley.
- Kruchten, P. (2004). Scaling down projects to meet the Agile sweet spot. *The Rational Edge.* http://www-106.ibm.com/developerworks/ rational/library/content/ RationalEdge/aug04/5558.html
- Kruchten, P. (2008). What do software architects really do*? Journal of Systems & Software*, 81(12), 2413-2416.
- Madison, J. (2010). Agile-Architecture Interactions. *IEEE Software,* 27(2), 41-47.
- McConnell, S. (2007). Technical Debt. 10x Software Development [cited 2010 June 14]; http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx.
- Mills, J. A. (1985). A Pragmatic View of the System Architect. *Comm. ACM*, 28(7), 708-717.
- Nord, R. L., & Tomayko, J. E. (2006). Software Architecture-Centric Methods and Agile Development. *IEEE Software*, 23(2), 47-53.



References (4)

- McConnell, S. (20087) Notes on Technical Debt, Blog post at: http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx
- Parsons, R. (2008). *Architecture and Agile Methodologies—How to Get Along.* Tutorial At WICSA 2008, Vancouver, BC.
- Qumer, A., & Henderson-Sellers, B. (2008). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology*, 50(4), 280-295.
- Rendell, A. (2009) "Descending from the Architect's Ivory Tower," in *Agile 2009 Conference,* A. Sidky, et al., eds. IEEE Computer Society, pp. 180-185.
- Rozanski, N., & Woods, E. (2005). *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.* Addison-Wesley.
- Special issue of Cutter IT Journal, edited by I. Gat (October 2010) Cutter IT Journal, 23 (10).
- Sterling, C. (2010) *Managing Software Debt*, Addison-Wesley.
- Wiegers, K. (1999). First Things First: Prioritizing Requirements. *Software Development Magazine*, 7(9), 48-53.
- Woods, E. (2010). *Agile Principles and Software Architecture*, presentation at OOP 2010 Conf., Munich, Jan 26.
- Saaty, T. (1990). How to make a decision: The analytic hierarchy process. European journal of operational research, 48(1), 9-26.