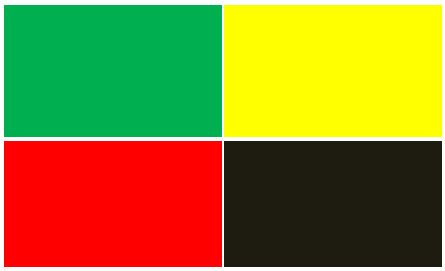


What do we do next?
Or: What colour is your backlog?

Philippe Kruchten
Scrum Gathering
Orlando, March 16-20, 2009

1

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

Professor of Software Engineering
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC Canada
pbk@ece.ubc.ca
+1 604 827-5654

Founder and president

Kruchten Engineering Services Ltd
Vancouver, BC Canada
philippe@kruchten.com
+1 604 418-2006

2

Outline

- Context
- Software development
- Backlog
- Time-box
- Features &Value
- Work & Cost
- Invisible features
- Dependencies
- Release and budget
- Dollars, or points & utils
- Estimation
- Buffers
- Defects
- Technical debt
- Constraints
- Time and depreciation
- Tool support
- Colours
- Research agenda
- Recommendations

3

No more planning

© Scott Adams, Inc./Dist. by UFS, Inc.

4

Context

- Research on modern software development practices
- Software project management
- "Evidence-based software-engineering"
- Old stuff in new clothes ?
- Integration

- *Financial support from Scrum Alliance*

5

A Conceptual Model of Software Development

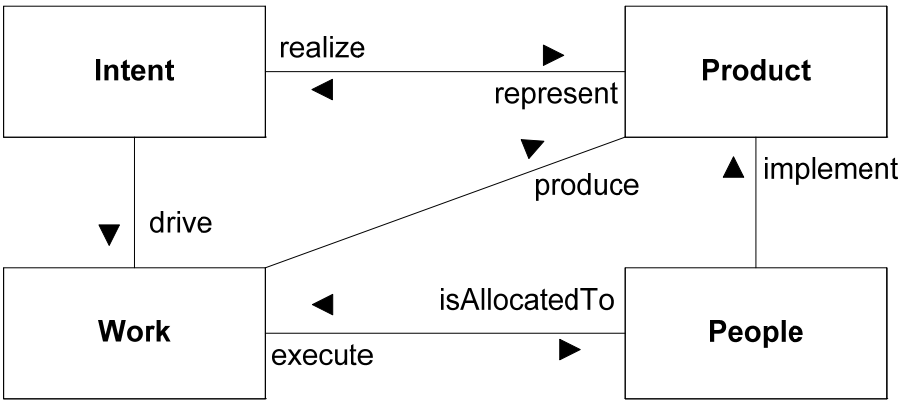
4 key concepts, 3 key attributes

- Intent
- Product
- Work
- People

- Time
- Quality
- Risk

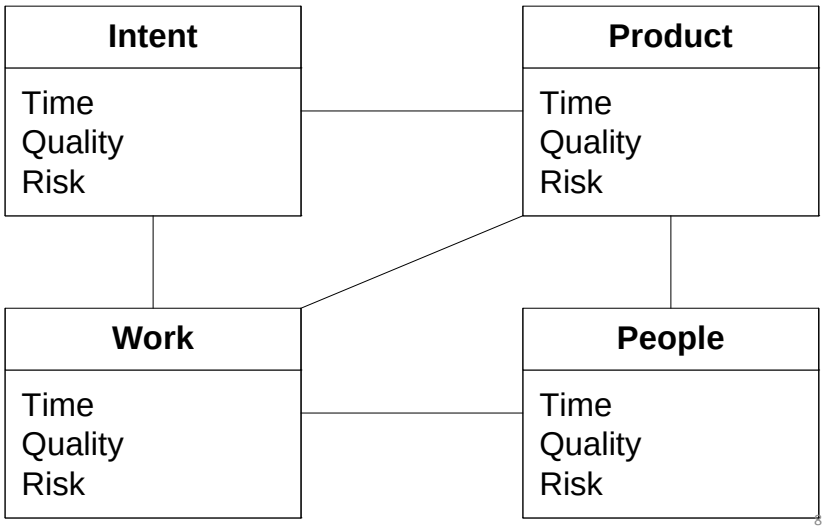
6

Intent, Work, People, Product

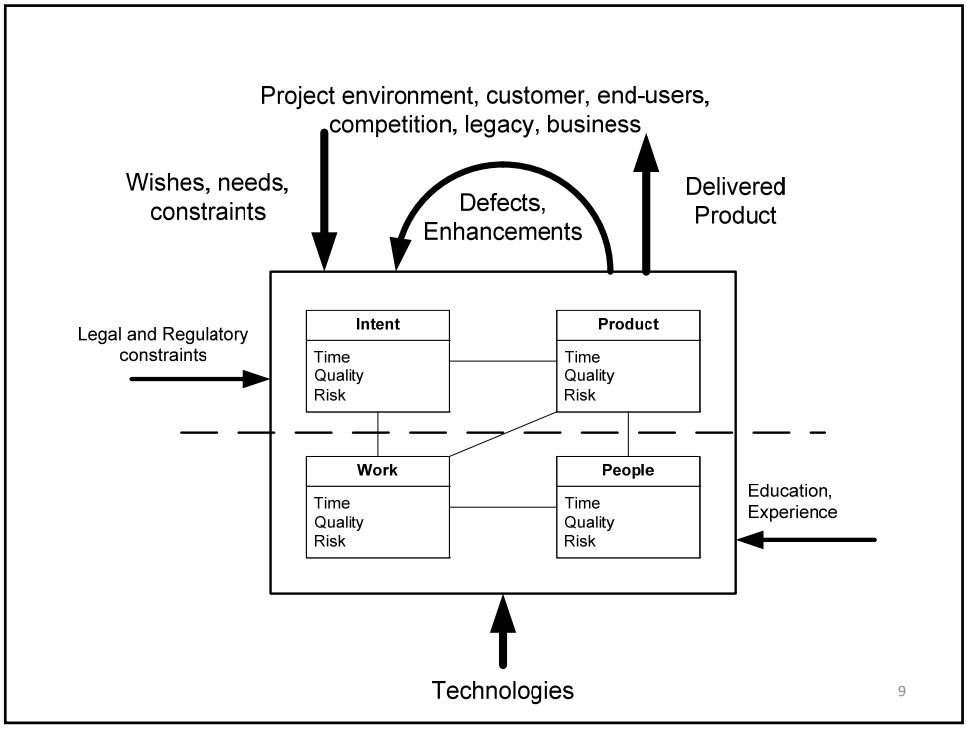


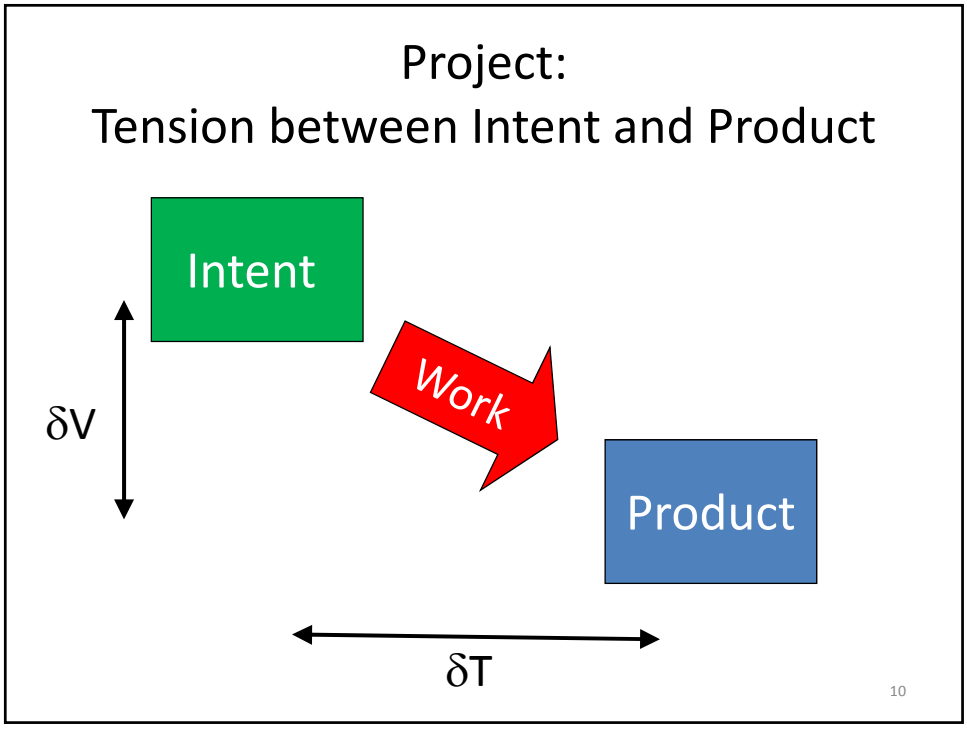
7

Adding Time, Quality & Risk

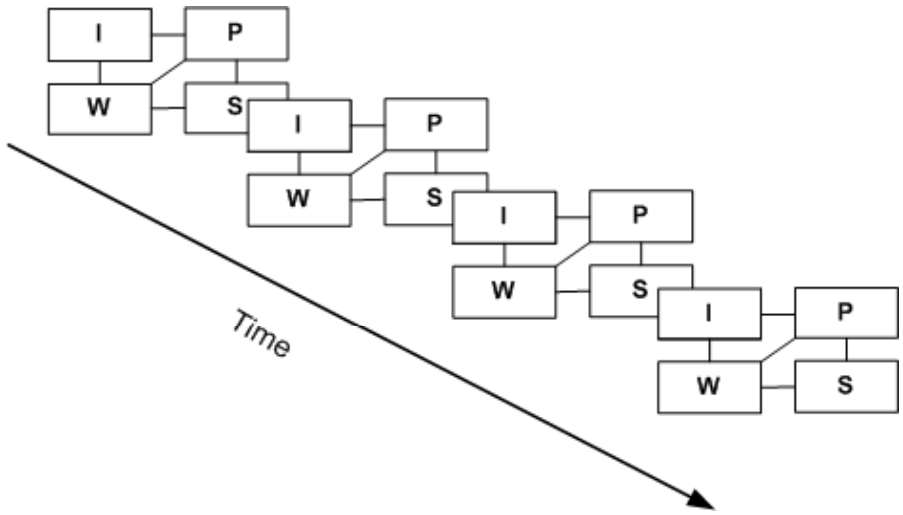


8


Project environment, customer, end-users, competition, legacy, business
Wishes, needs, constraints
Defects, Enhancements
Delivered Product
Legal and Regulatory constraints
Intent
Time
Quality
Risk
Product
Time
Quality
Risk
Work
Time
Quality
Risk
People
Time
Quality
Risk
Education, Experience
Technologies
9

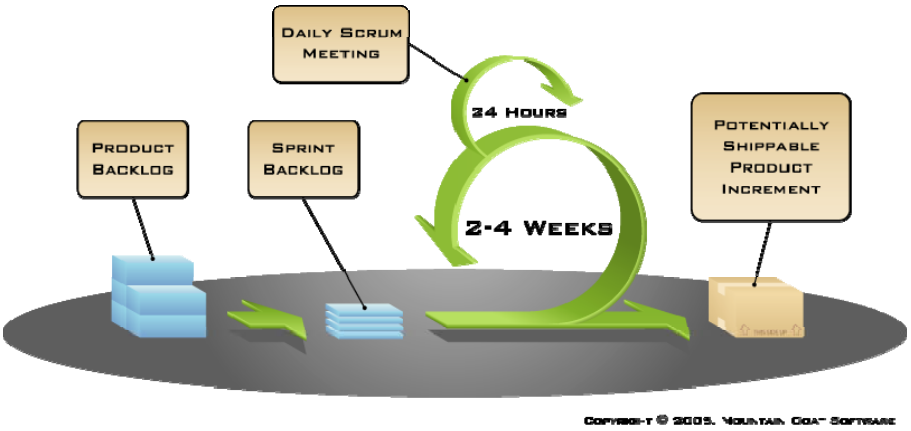

Project:
Tension between Intent and Product
Intent
Work
Product
δV
δT
10

Iterations

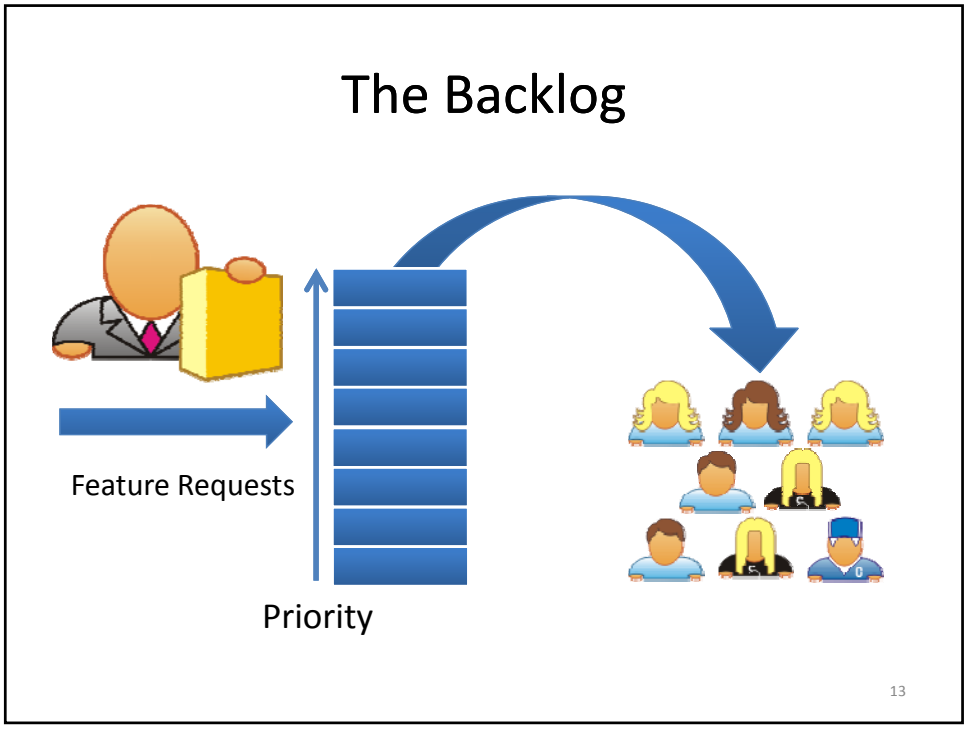


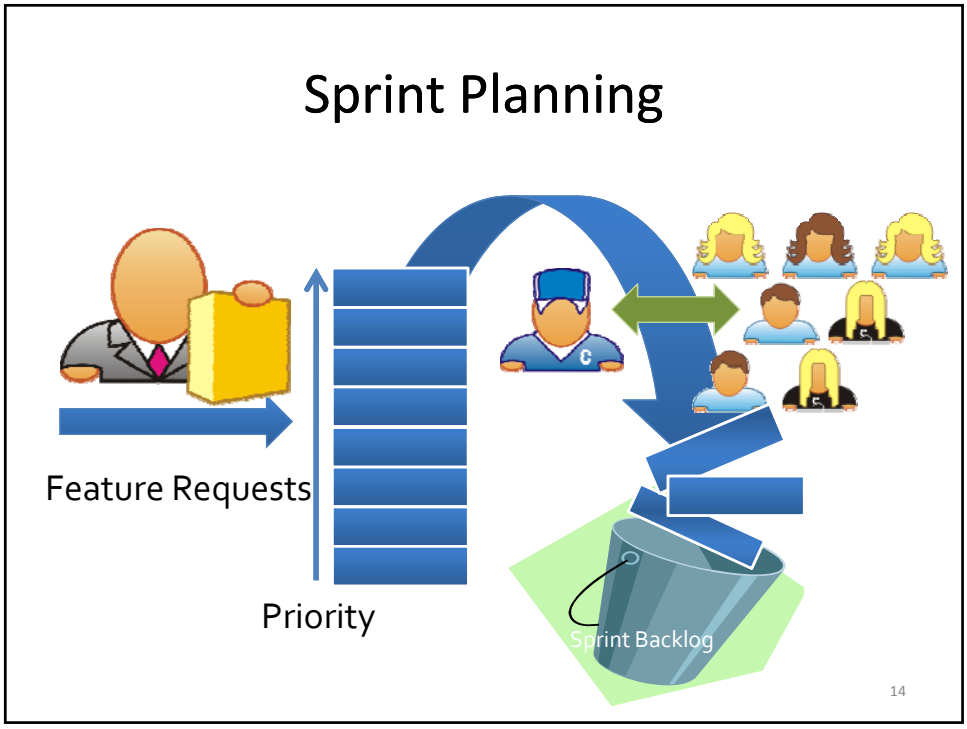
11

Backlog



12



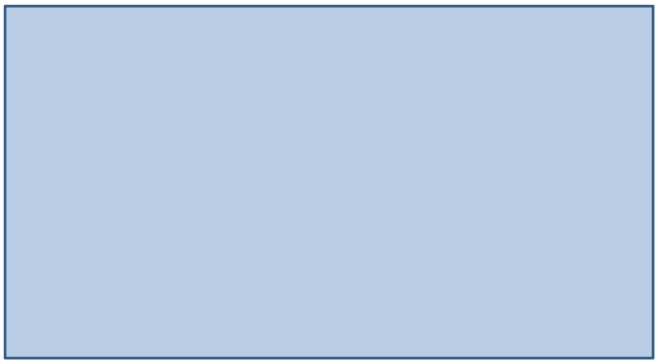


Painting your backlog

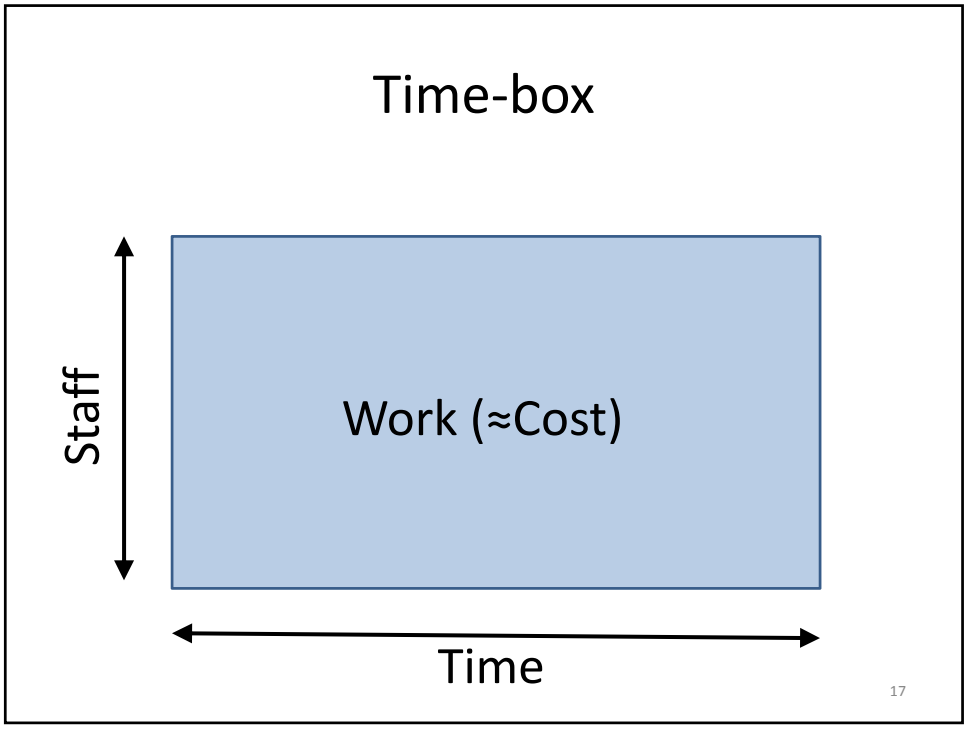
Features	Architecture infrastructure
Defects	Technical Debt

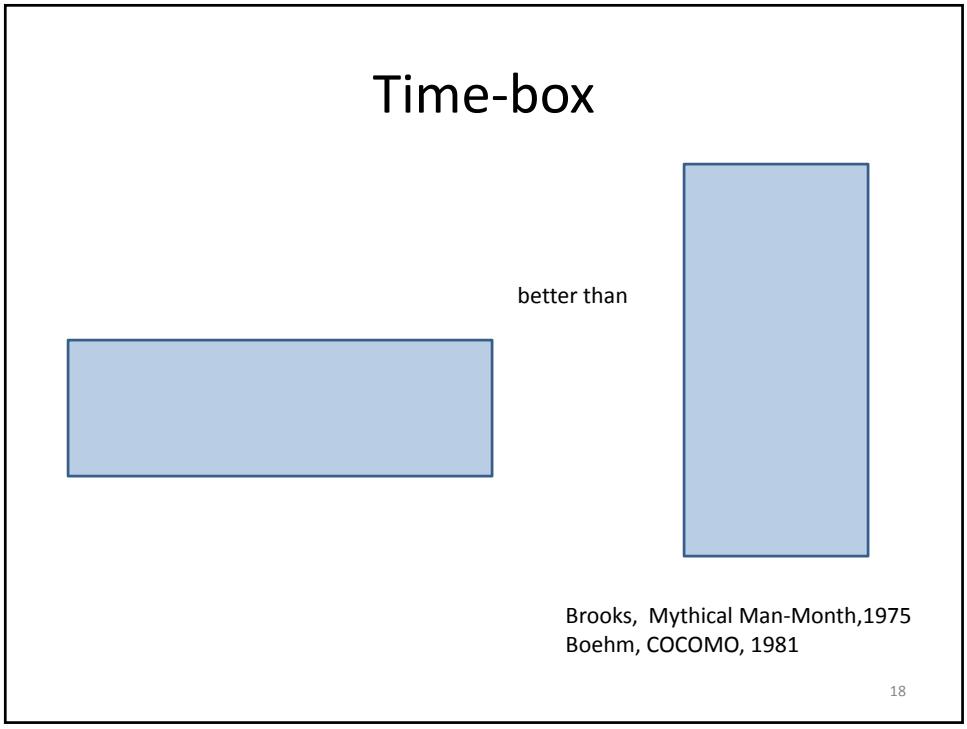
15

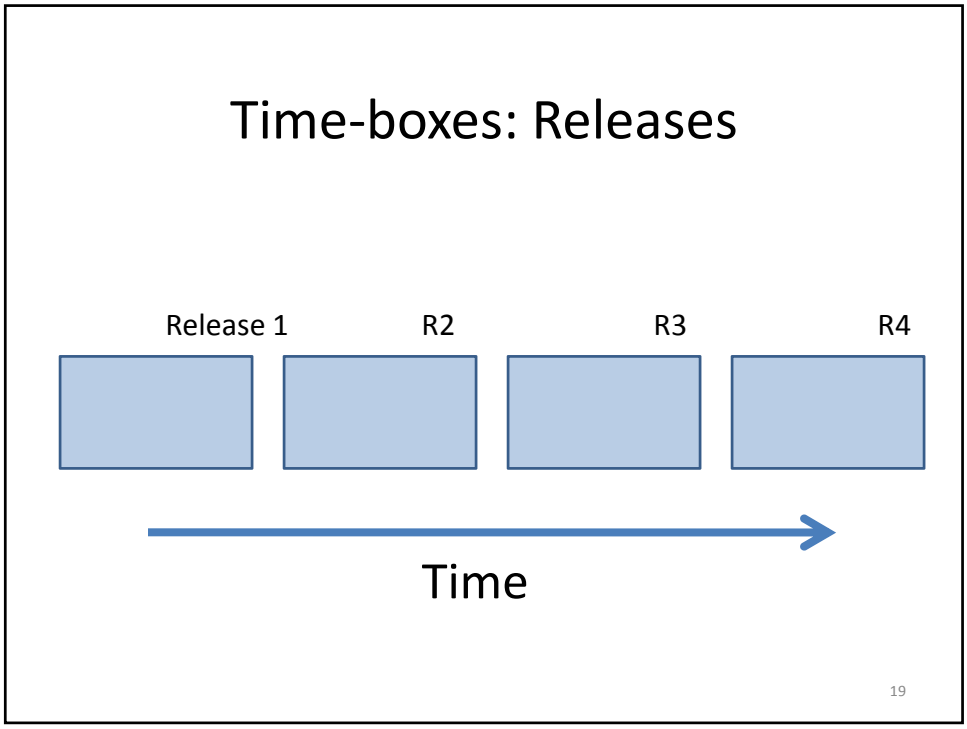
Time-box

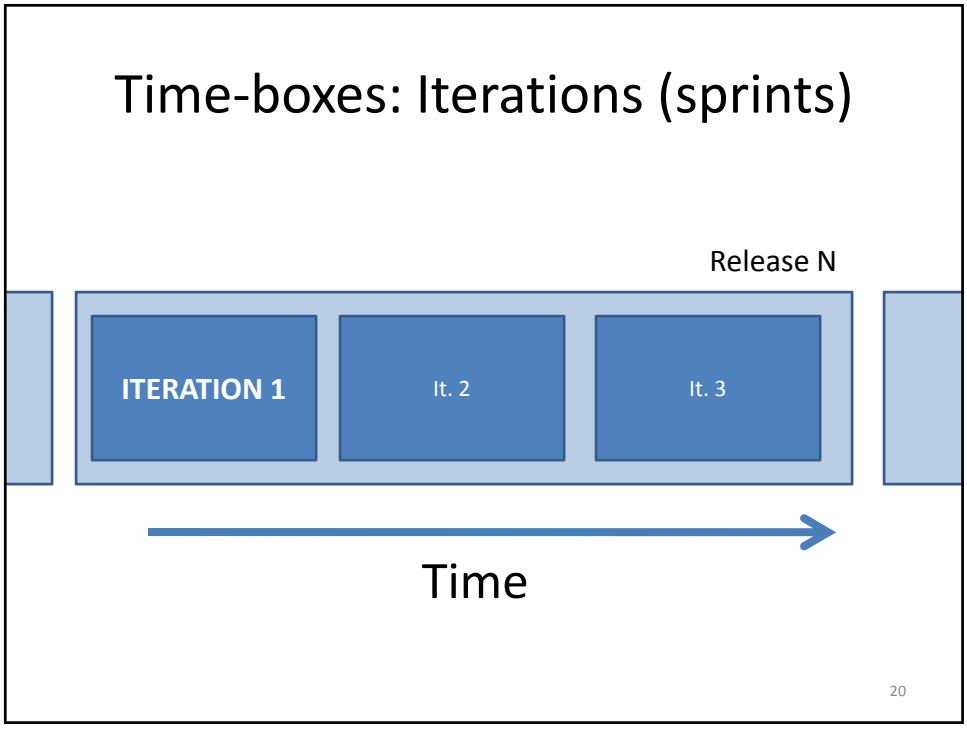


16

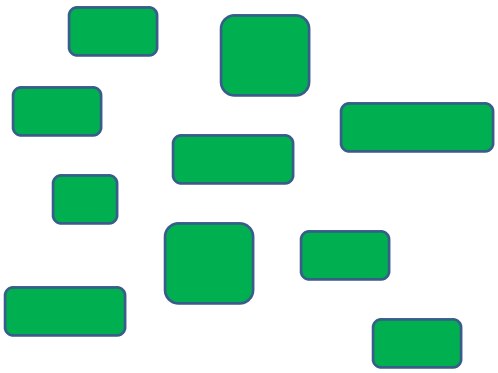






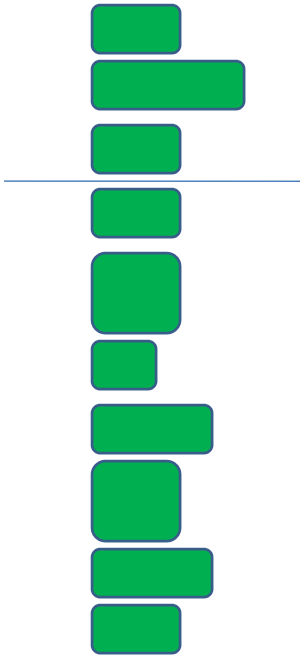


Features



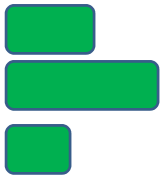
Intent

21

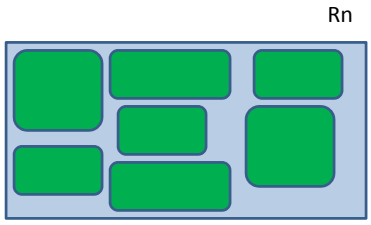


Features

22



Features

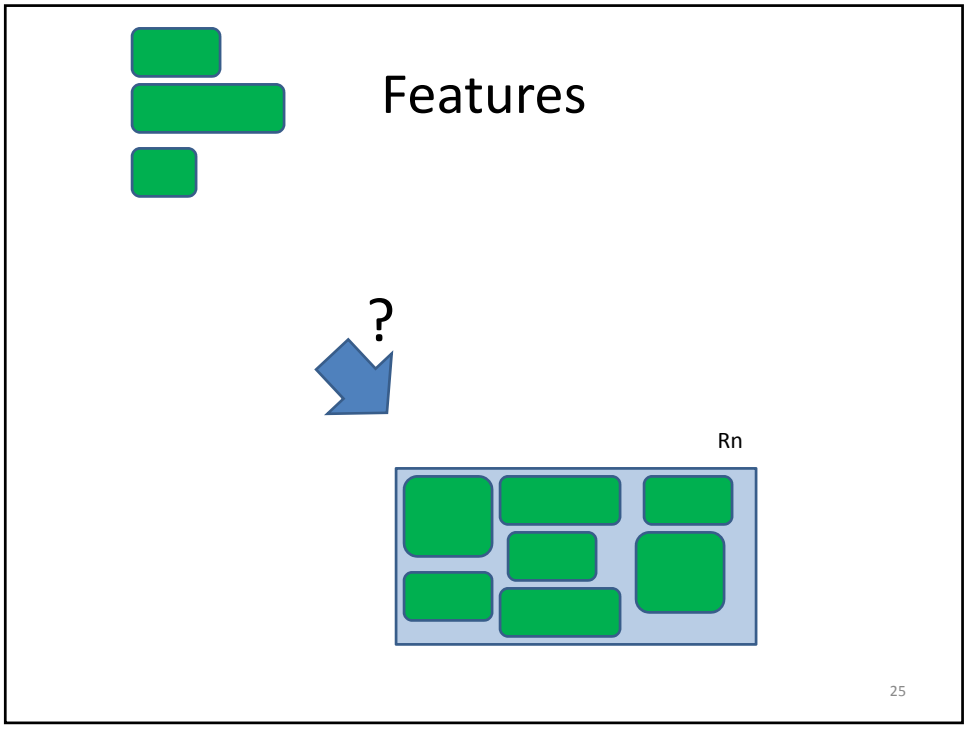


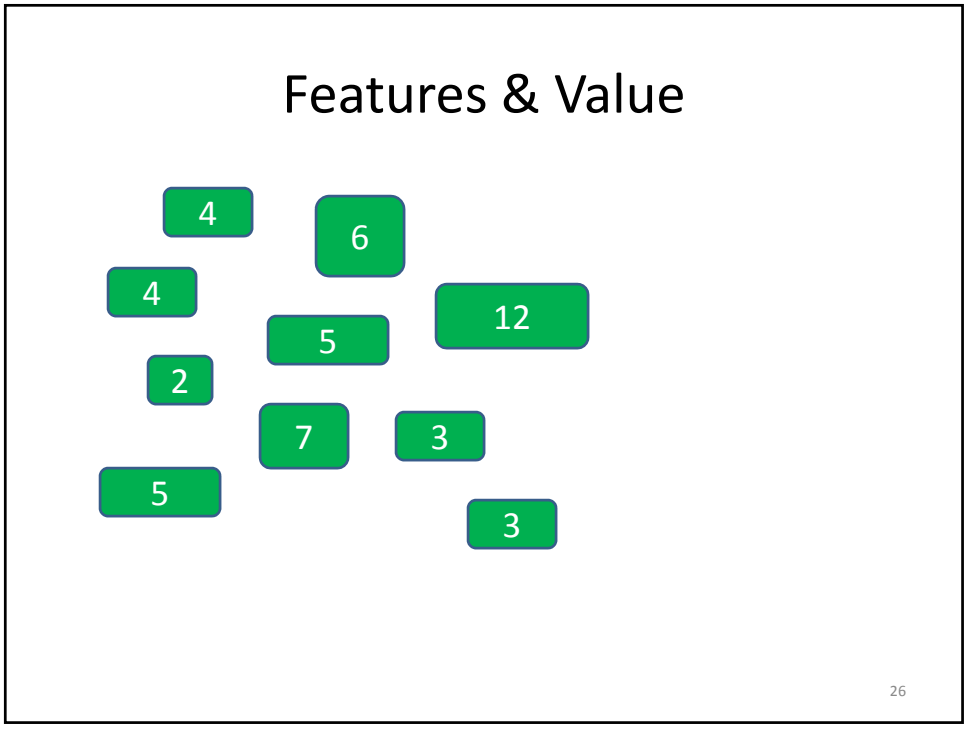
23

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?

24



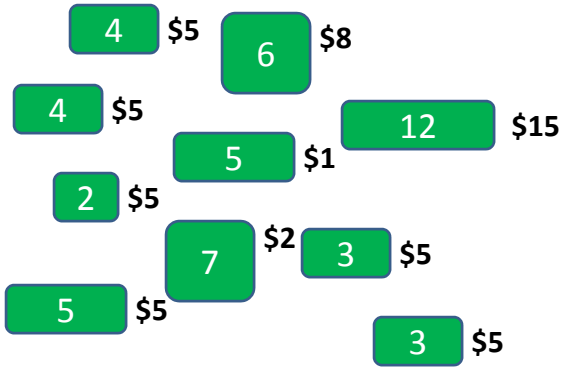


Maximizing value

Highest value first
Ignore time

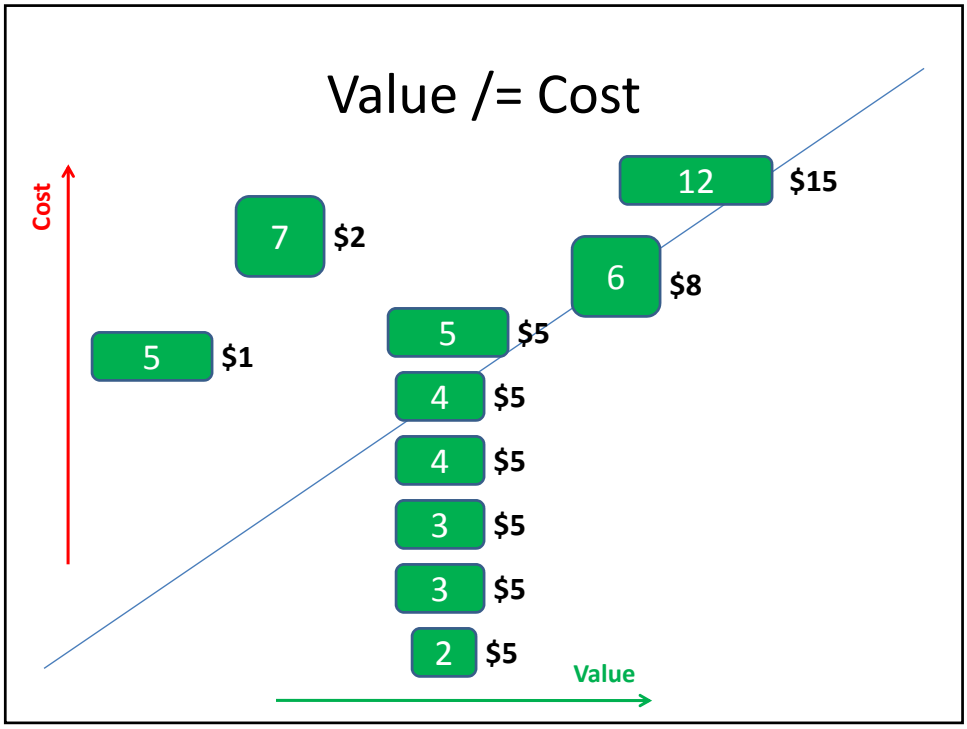
27

Value = Cost?



Only for simplest cases

28

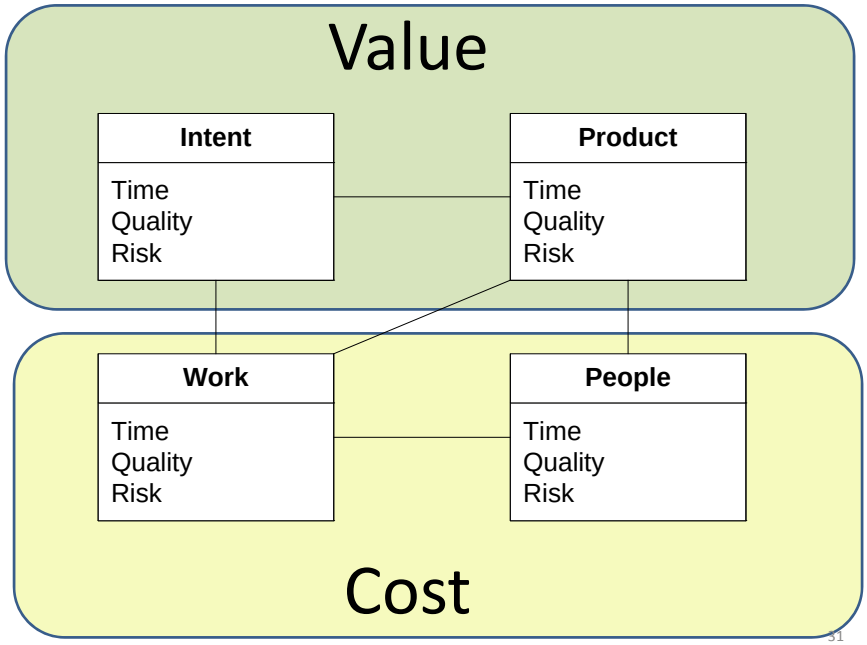


Value and Cost

- 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 ?

30



31

Efficiency *vs.* Effectiveness

Efficiency

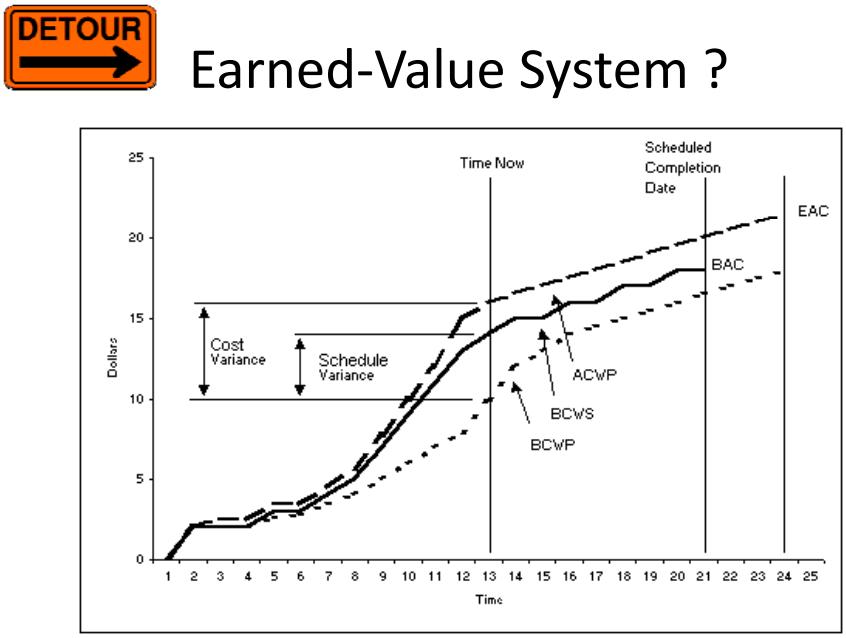
- relationship between the output in terms of goods, services or other results and the resources used to produce them

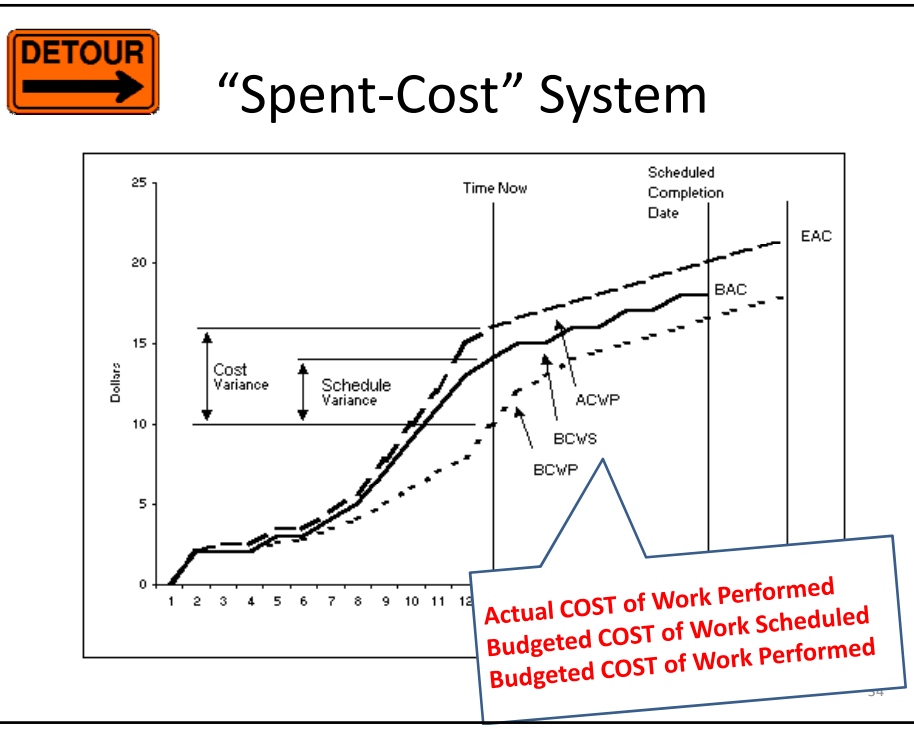
Effectiveness

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

Cost **Value**

32


DETOUR
Earned-Value System ?
Time Now
Scheduled Completion Date
EAC
BAC
Cost Variance
Schedule Variance
ACWP
BCWS
BCWP
Dollars
Time
33


DETOUR
"Spent-Cost" System
Time Now
Scheduled Completion Date
EAC
BAC
Cost Variance
Schedule Variance
ACWP
BCWS
BCWP
Dollars
Actual COST of Work Performed
Budgeted COST of Work Scheduled
Budgeted COST of Work Performed

Invisible Features

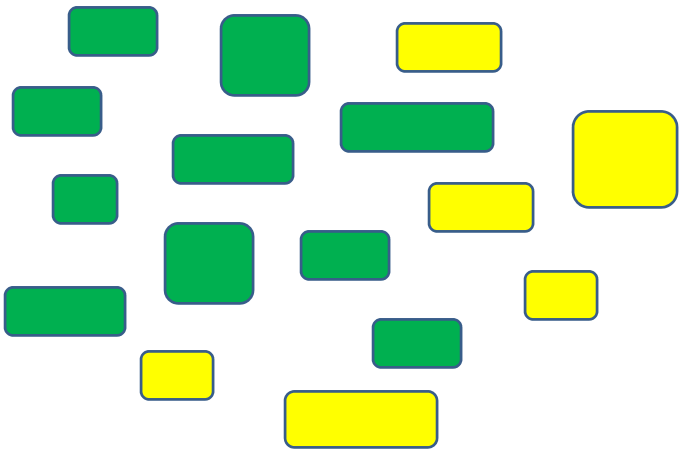
35

Invisible Features

- Architecture
- Infrastructure
- Common elements
- Libraries
- Reuse

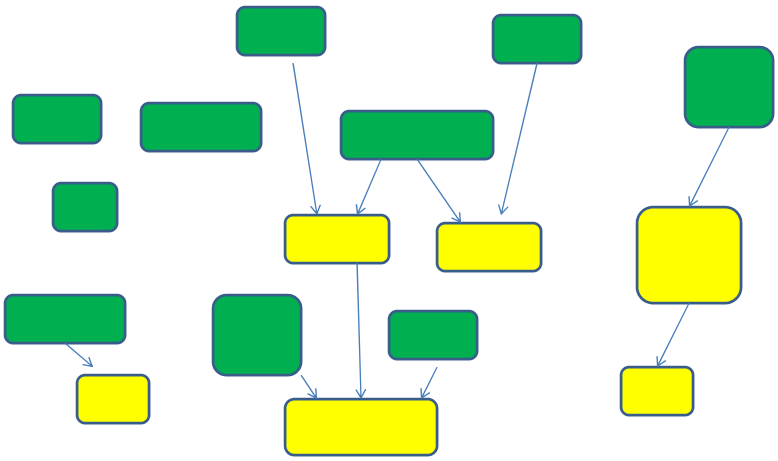
36

Features



37

Dependencies



38

Release Planning

- Time-box = budget
- Fill the time-box with a combination of visible and invisible features
- … while maximizing value
- Product manager: maximize value (green stuff)
- Project manager: maximize budget utilization
- Techie: maximize the fun stuff (yellow) ?

39

Units of Cost and Value

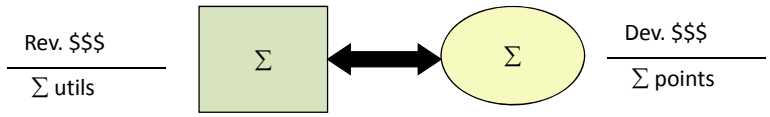
- Value in Dollars ?
  - Increases confusion value vs. cost
  - Very hard to define
- Priority
  - High, medium, low
  - MoSCoW
- Relative index
- Util

- Matches “points” for cost

43

Utils & Points

- Value
  - Measured in Utils
- Cost ≈ effort
  - Measured in Points



Bass et al 2003
Rick Kazman, SEI

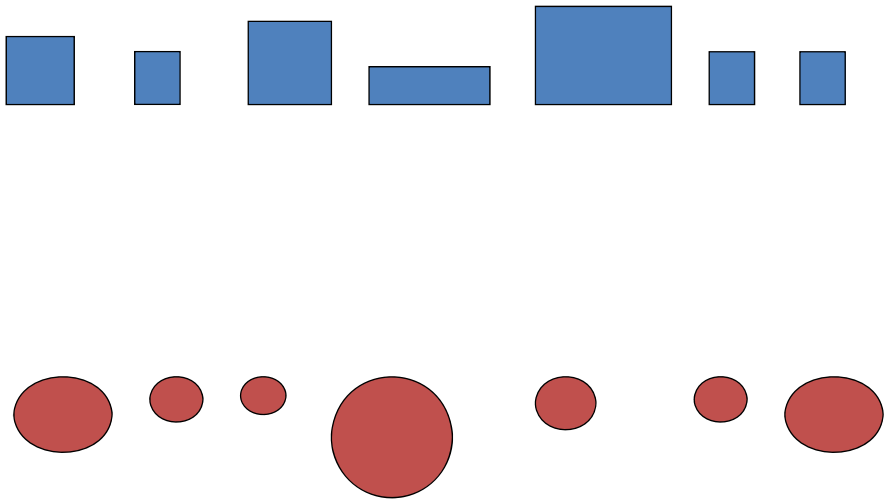
44

Research: Value "Flow down"

- Heuristics to allocate value to invisible features
- Assign value to visible features (utils)
- "Borrow" value from visible features and allocate to invisible features, using dependencies
- Keep total value constant
- Goal: using value and dependencies to sequence development

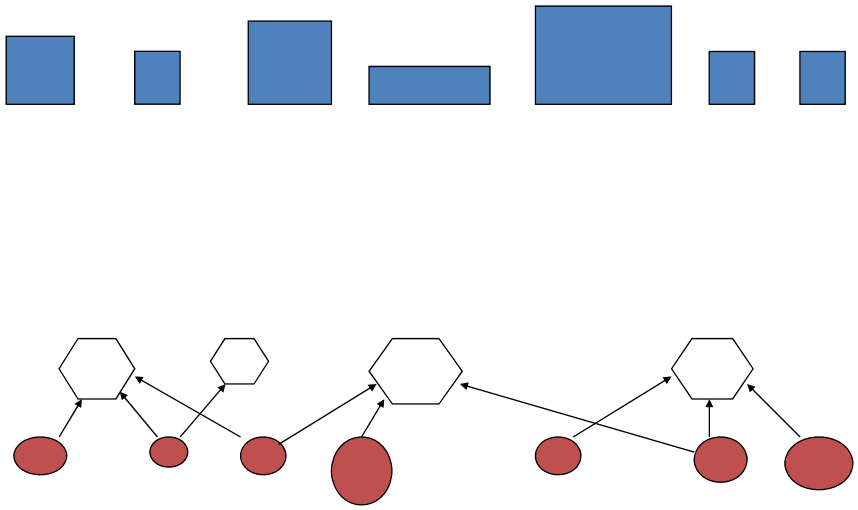
45

Points (cost) and Utils (value)



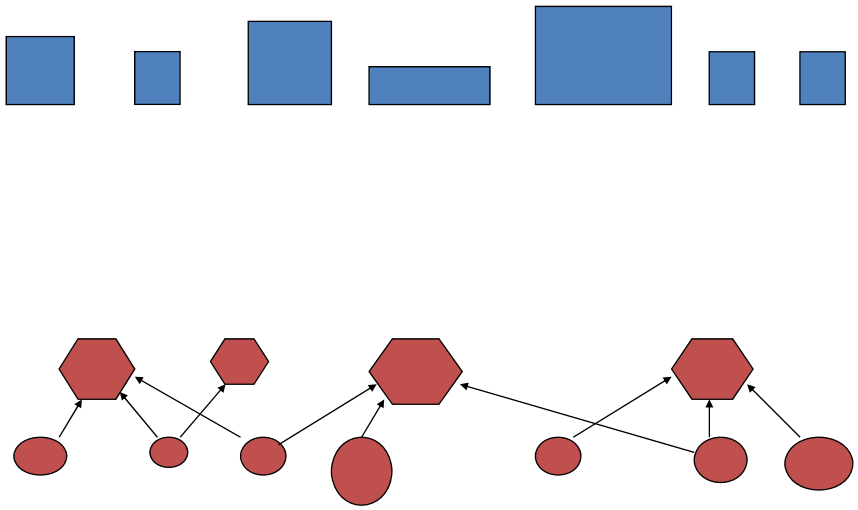
47

Points (cost) and Utils (value)



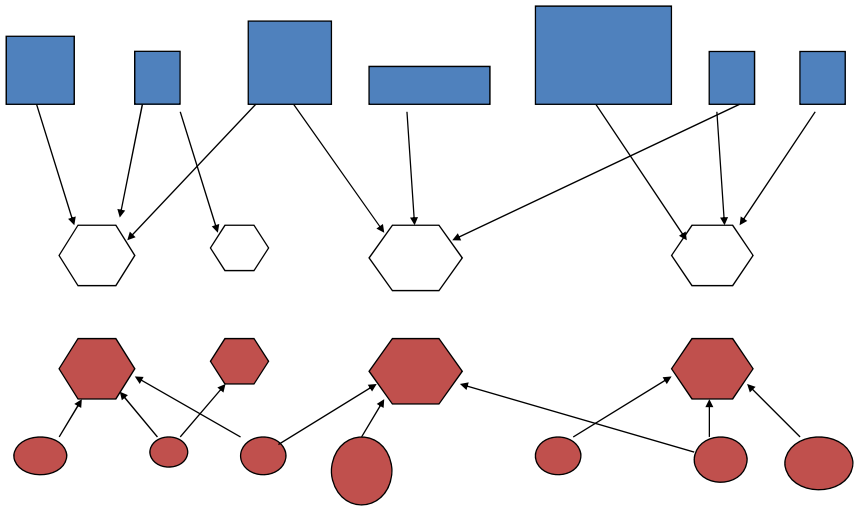
49

Points (cost) and Utils (value)



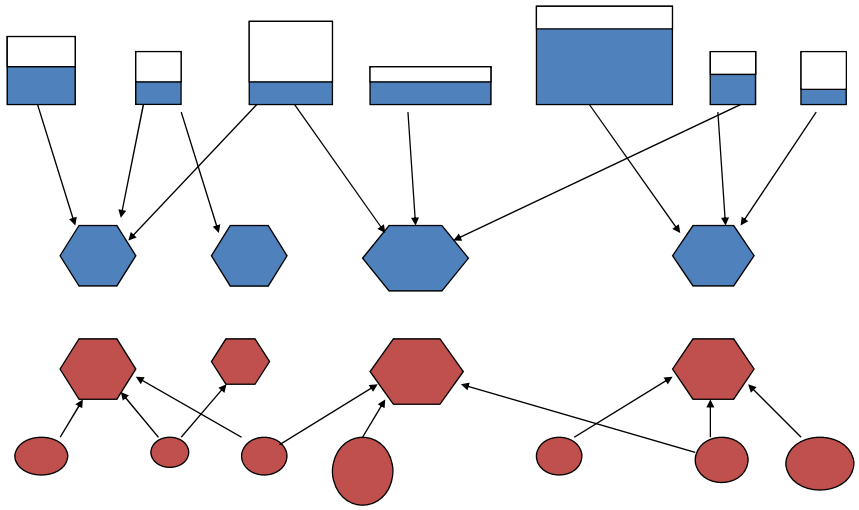
50

Points (cost) and Utils (value)



52

Points (cost) and Utils (value)



53

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)

54

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*

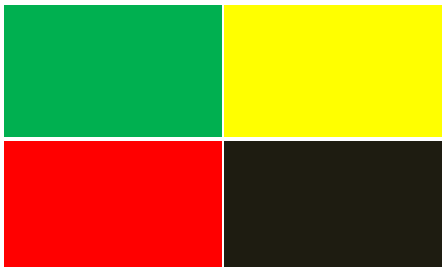
- Analytic Hierarchy Process (AHP) Saaty, 1990
- Evolve* - Hybrid
  - Günther Ruhe & D. Greer 2003, etc…

55

IMF: 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

56



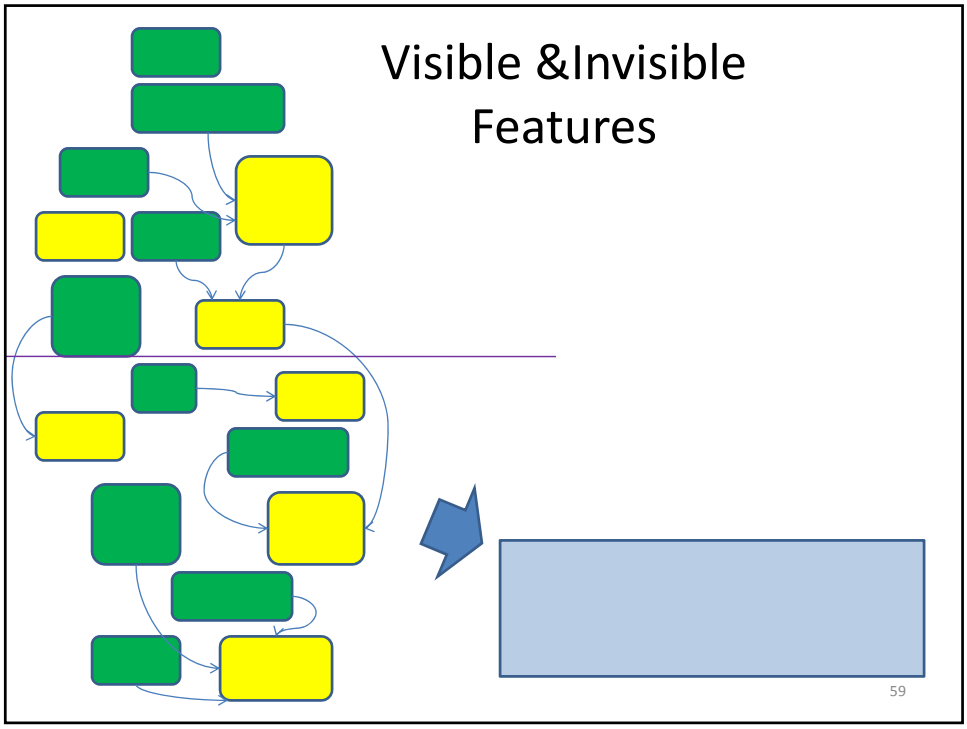
What colour is your backlog?

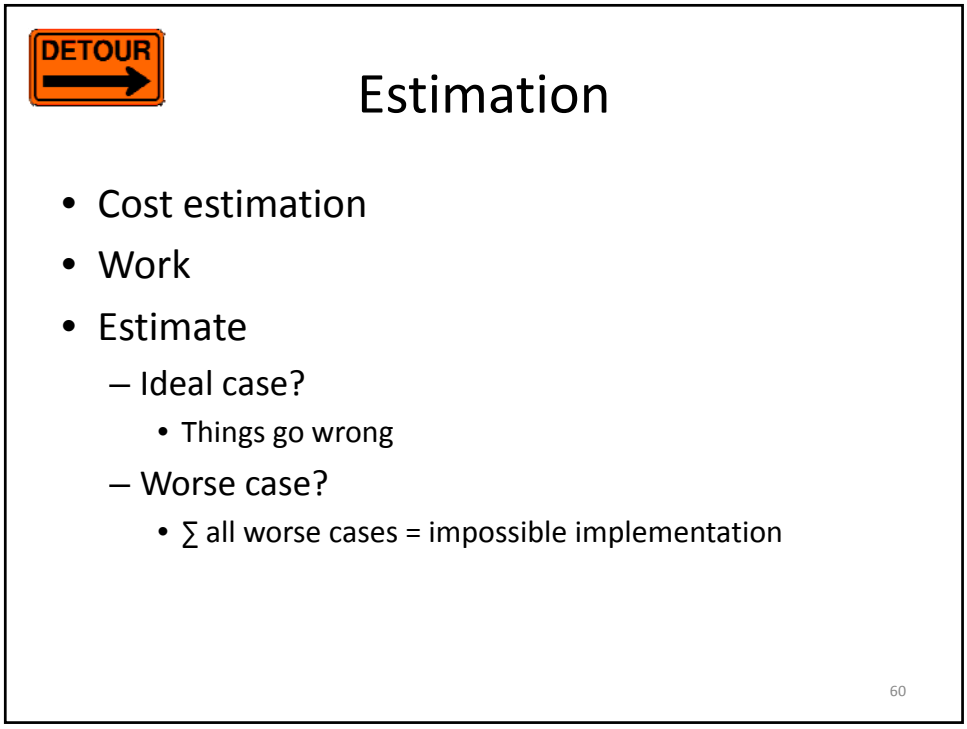
(so far)

57

Features

58


Visible &Invisible
Features
59


DETOUR
Estimation
• Cost estimation
• Work
• Estimate
– Ideal case?
• Things go wrong
– Worse case?
• ∑ all worse cases = impossible implementation
60

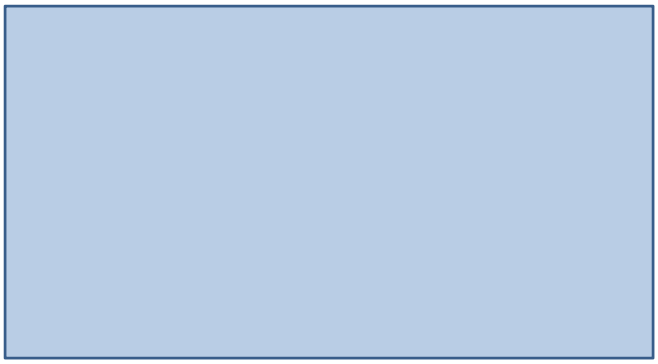
Buffers

- E. Goldratt: Theory of constraints
- D. Anderson

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

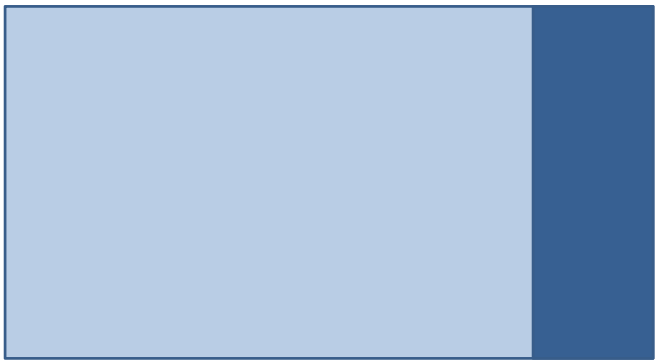
61

Time-box



62

Time-box with Buffer



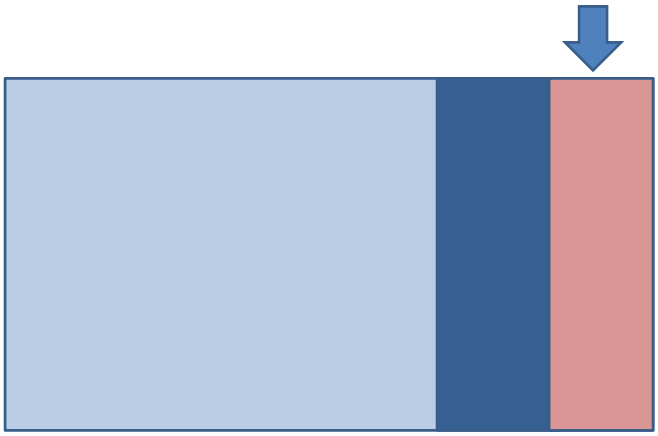
63

Defects

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

64

Buffer for in-process defect



65

Defect Value

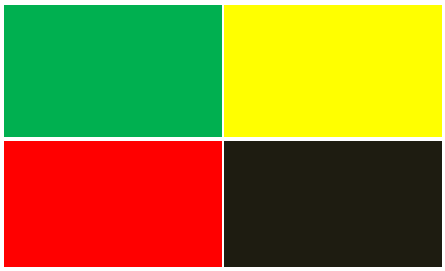
66

Fixing a Defect

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

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

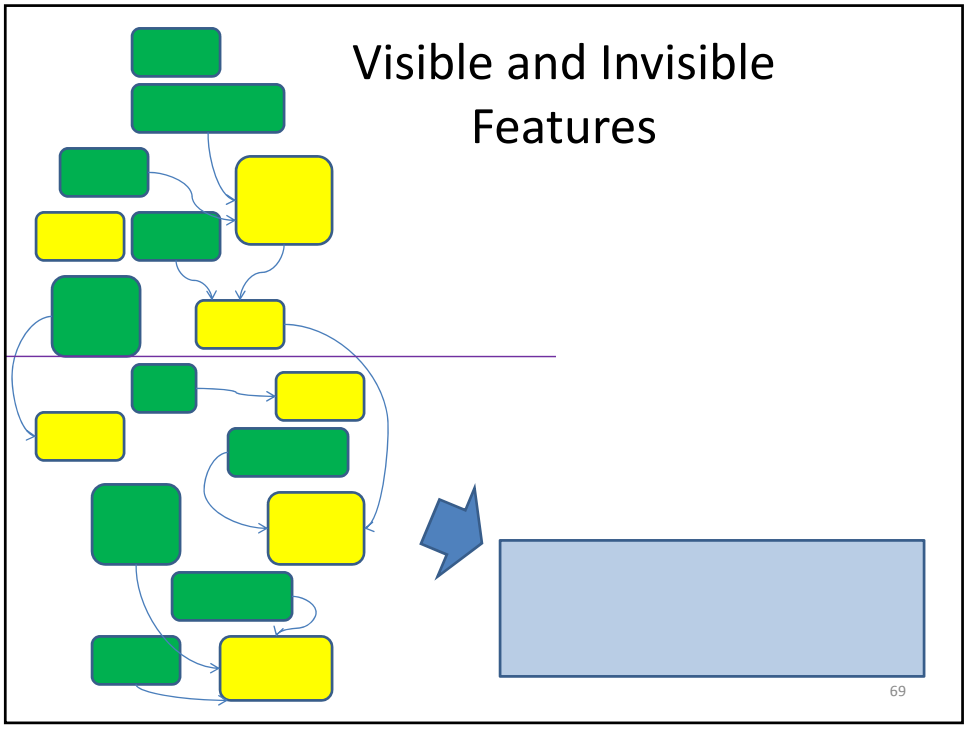
67

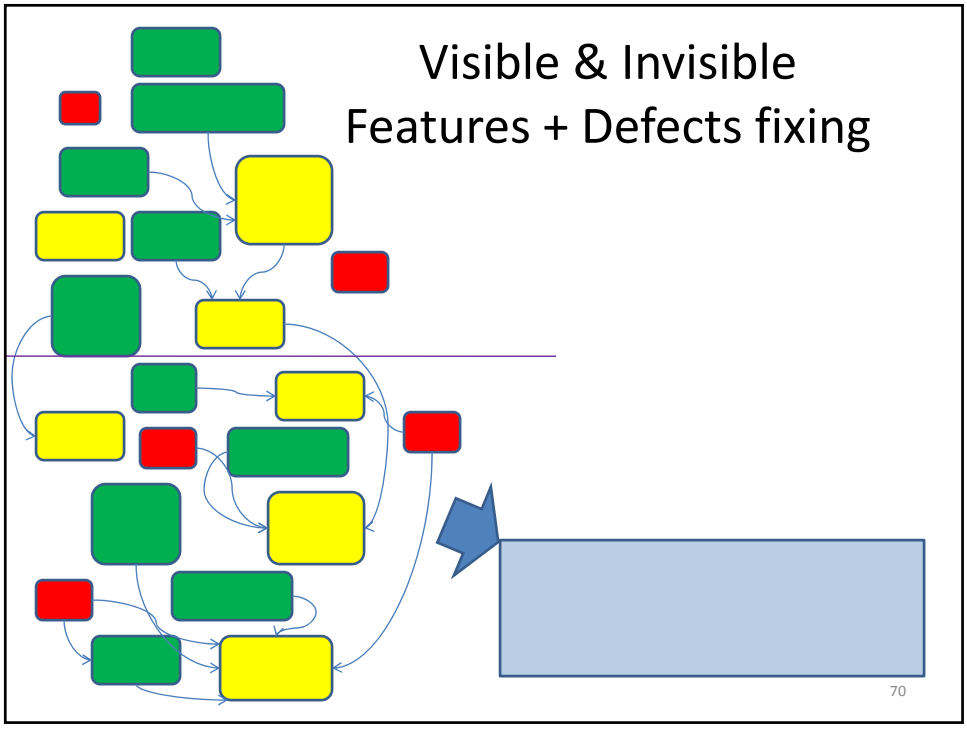


What colour is your backlog?

(so far)

68





Technical Debt

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

Cunningham, OOPSLA 1992

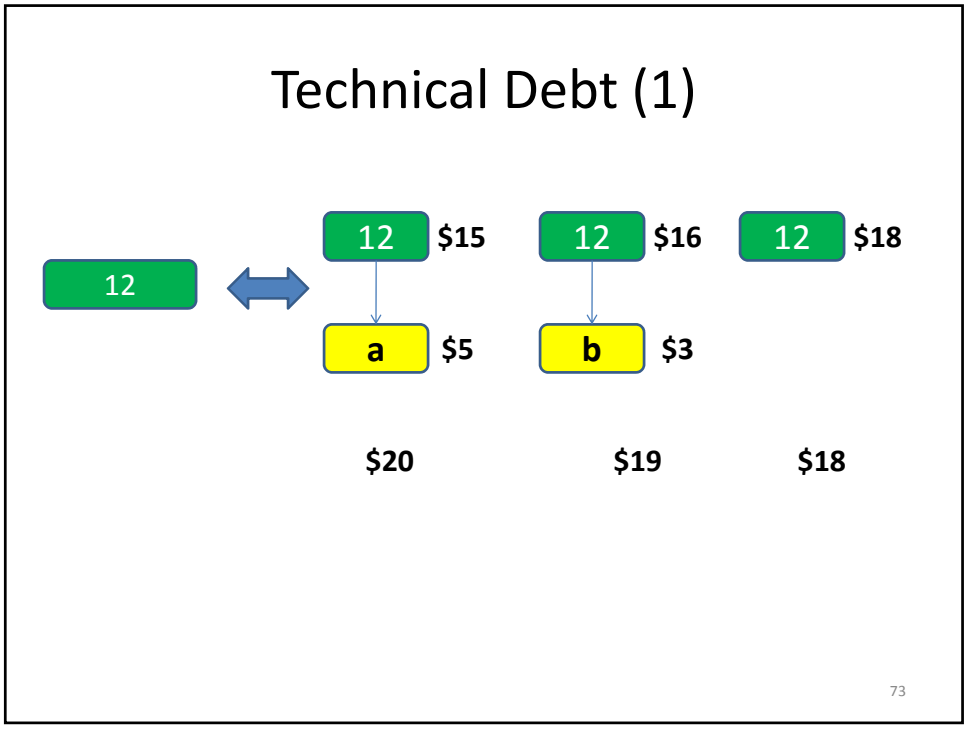
71

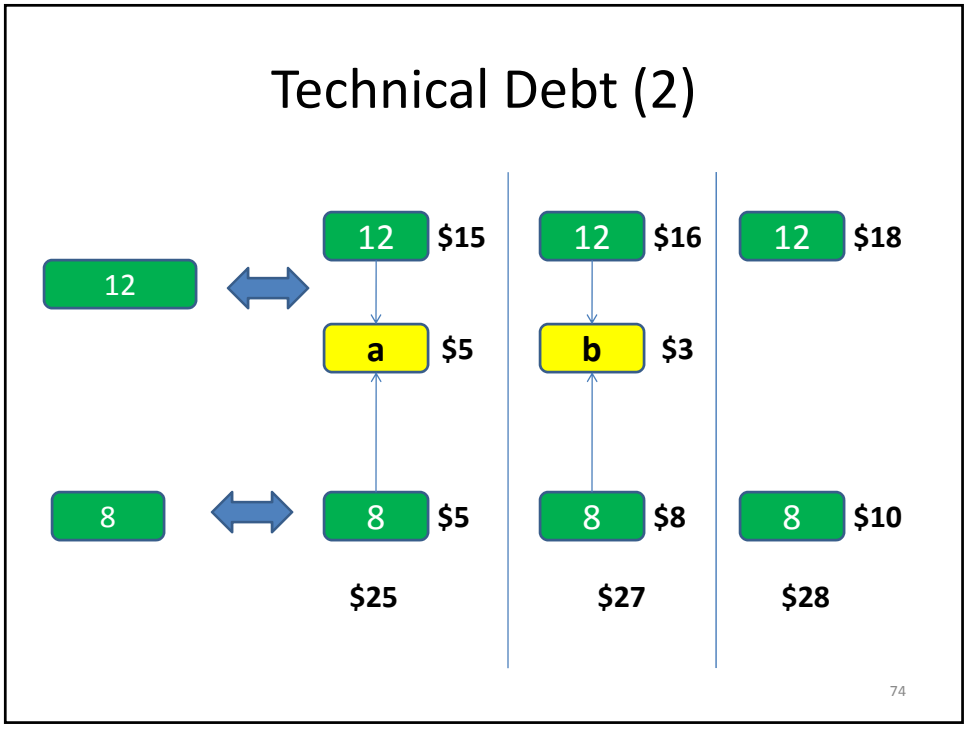
Technical Debt (S. McConnell)

- 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

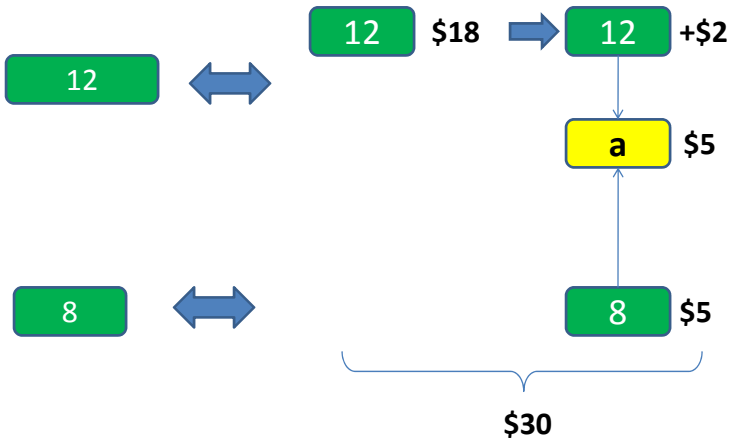
McConnell 2007

72


Technical Debt (1)
12
12
$15
a
$5
12
$16
b
$3
12
$18
$20
$19
$18
73


Technical Debt (2)
12
12
$15
a
$5
12
$16
b
$3
12
$18
8
8
$5
8
$8
8
$10
$25
$27
$28
74

Technical Debt (3)



75

Technical Debt

- Defect = Visible feature with negative value
- Technical debt = Invisible “feature” with negative value

- Cost of fixing
- Value of repaying technical debt

76

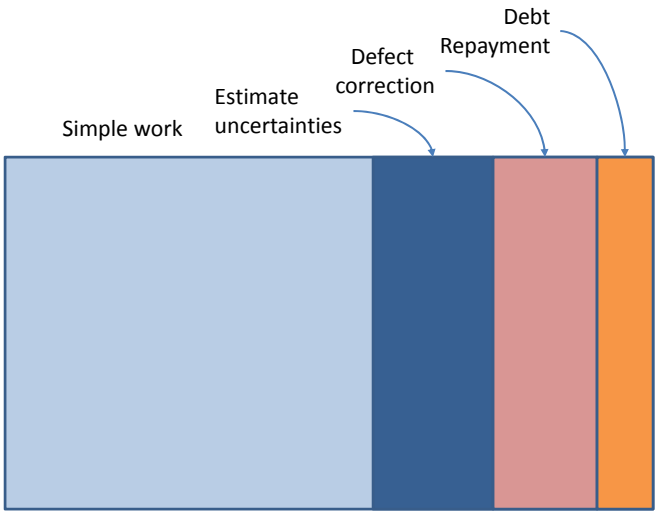
Interests

- In presence of technical debt
- Cost of adding new feature higher
- When repaying (fixing), additional cost for retrofitting already implemented features
- Technical debt not repaid => lead to increased cost, forever
- Cost of fixing increases

M. Fowler

77

Buffer for debt repayment



78

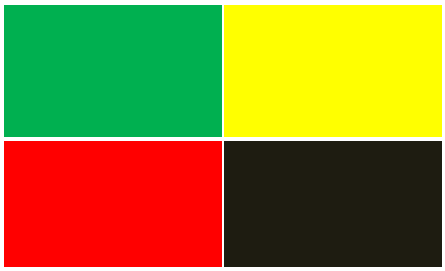
Colours in a Product

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

79

- 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.

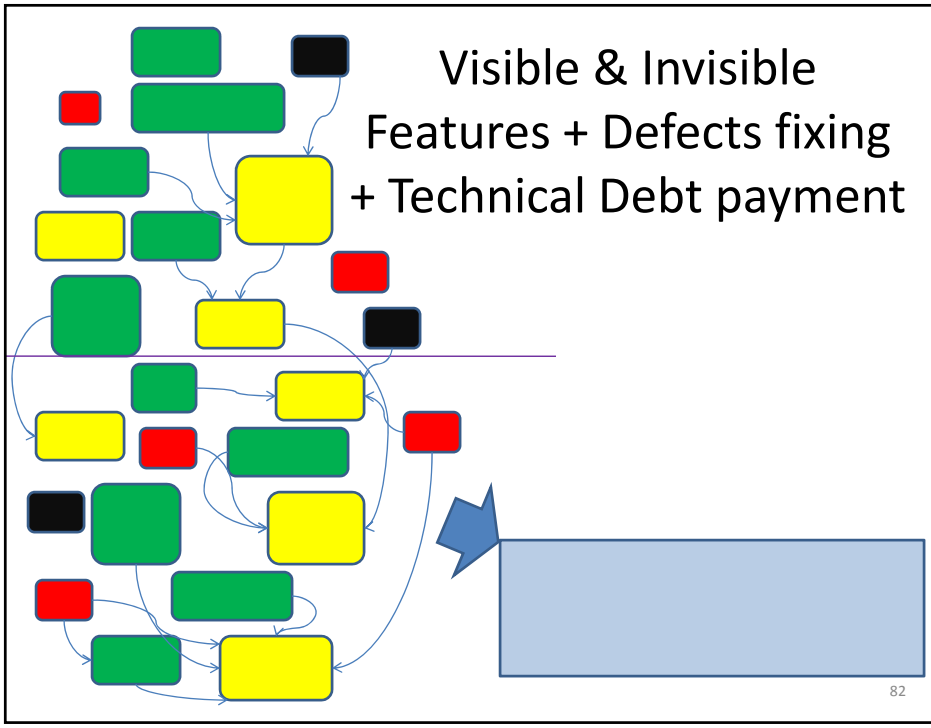
80



What colour is your backlog?

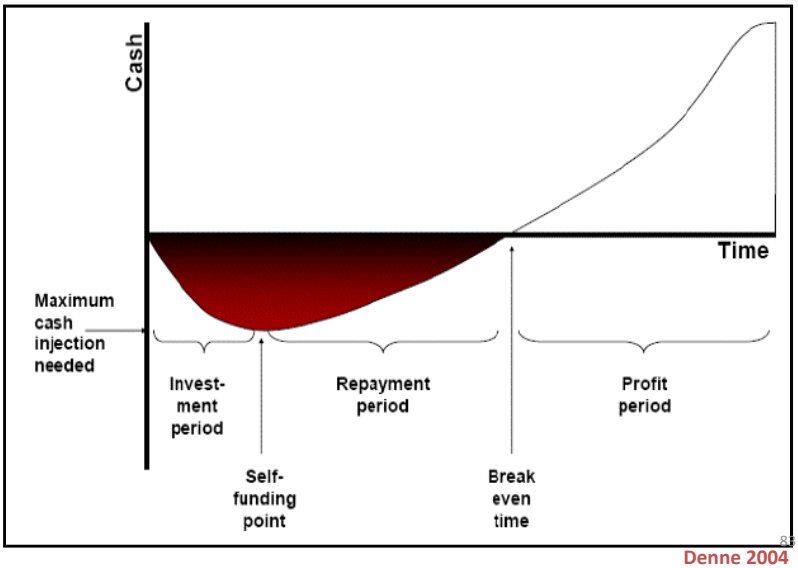
(so far)

81



82

Time and depreciation



Denne 2004

83

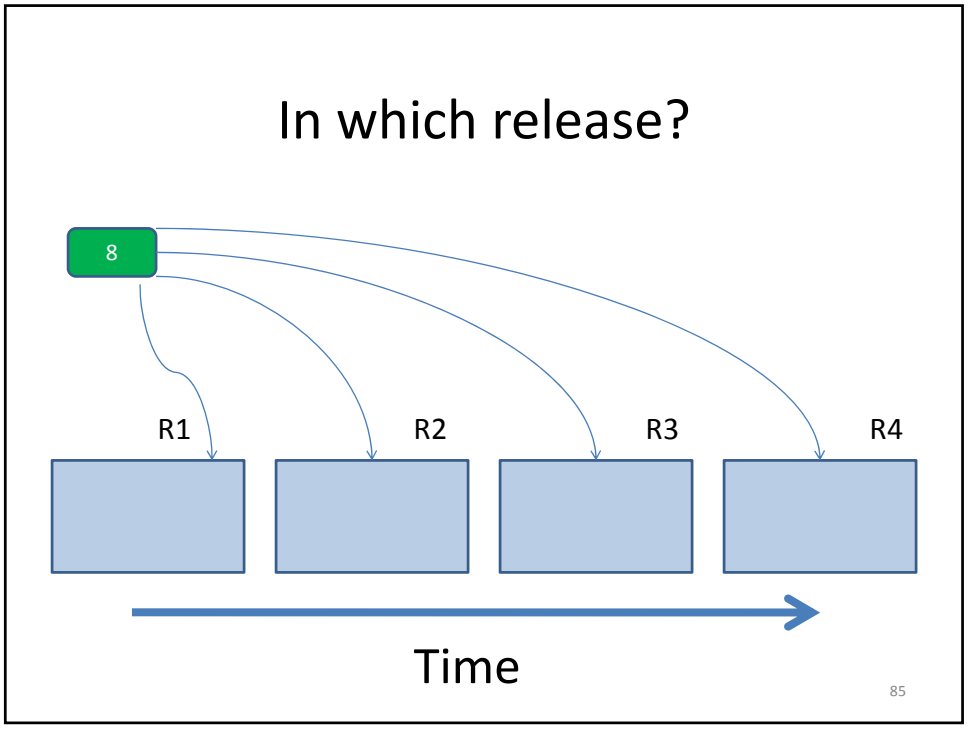
Net Present Value

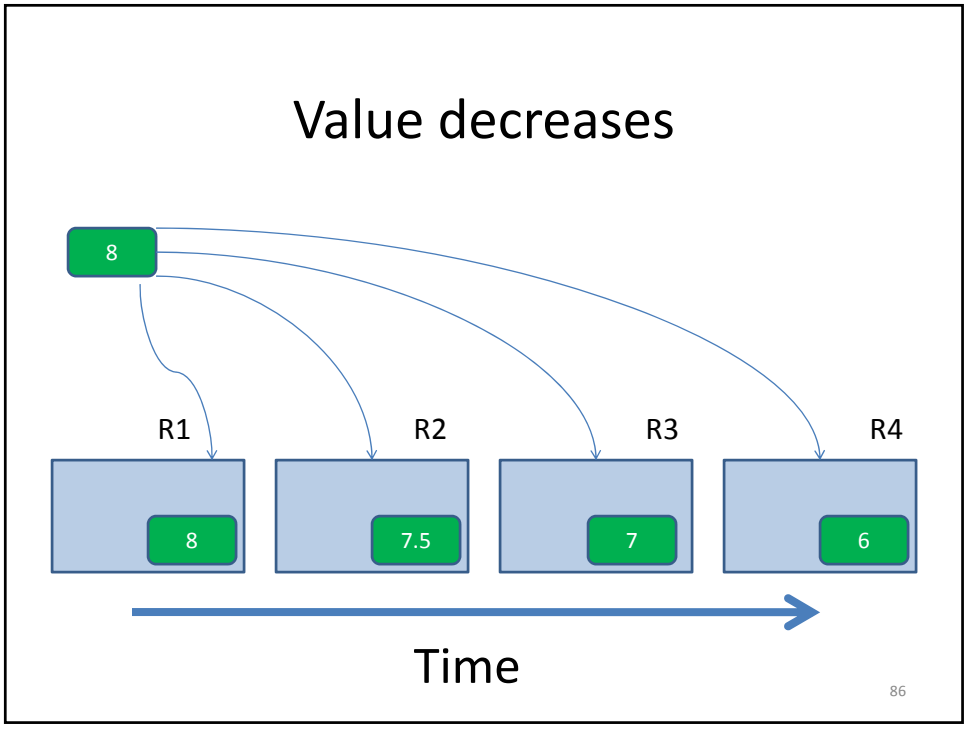
Net Present Value (NPV)

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

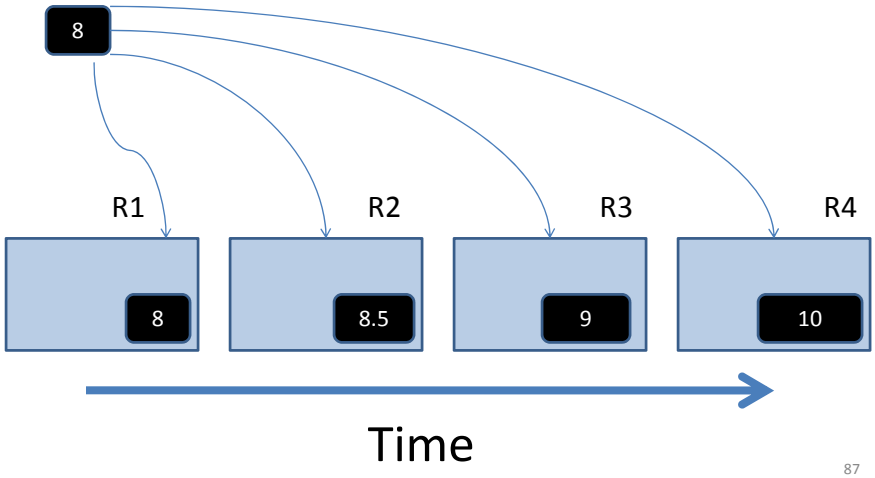
t = Cash Flow Period
i = Interest Rate Assumption

84





Technical debt: increase (?)

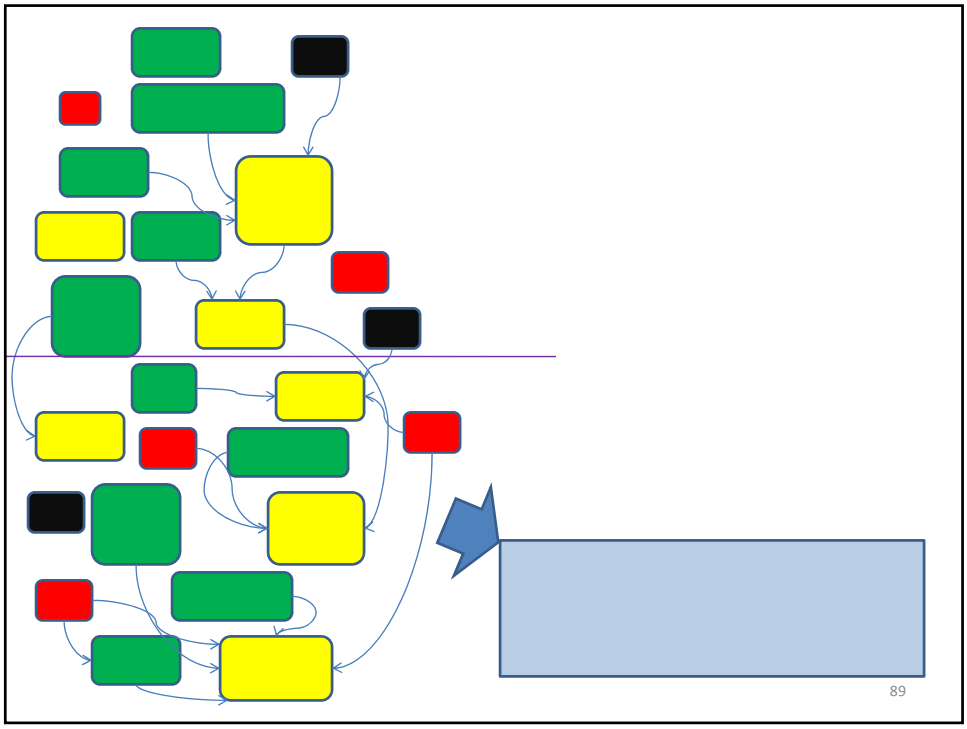


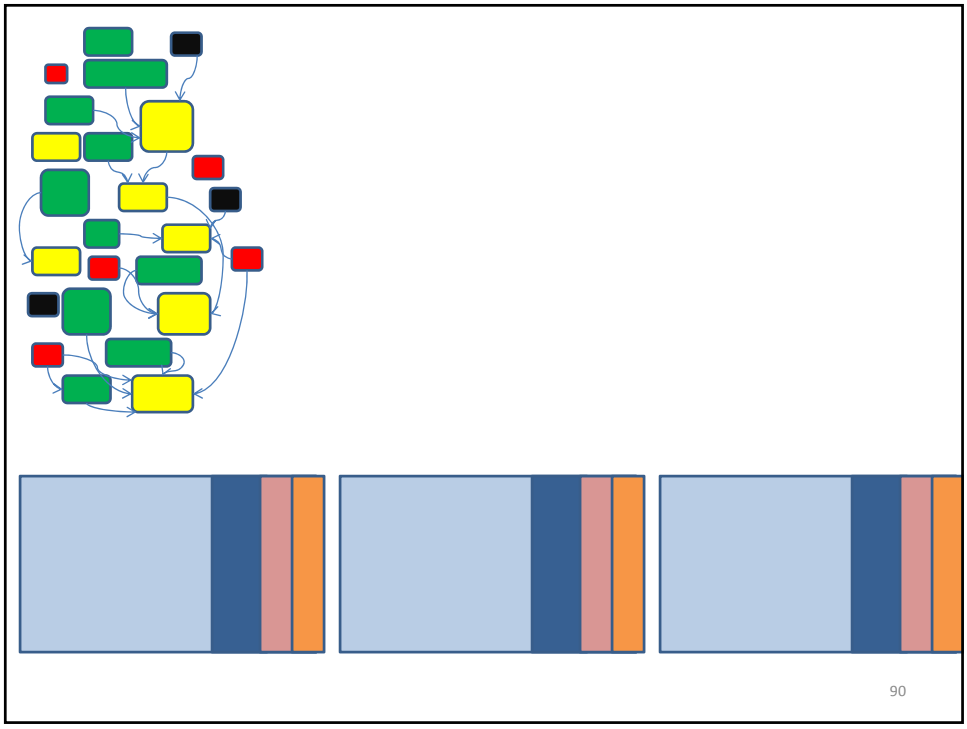
87

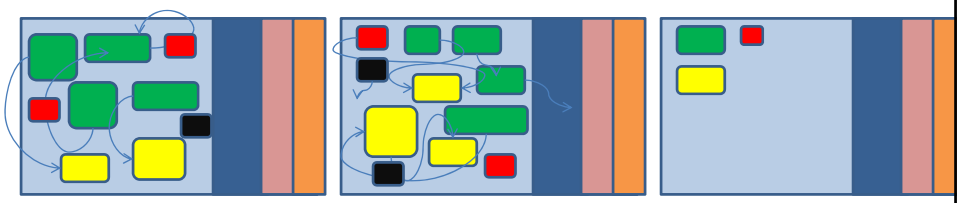
Tool support

- Strategic and tactical planning
  - Release and iteration (sprint) level
- Uses:
  - Value and cost
  - Dependencies
  - Depreciation
- Integrate concept of buffers
- Graphical UI (hence the colors)
- Add-on to existing scrum supporting tool
- Needed for experimentation and validation

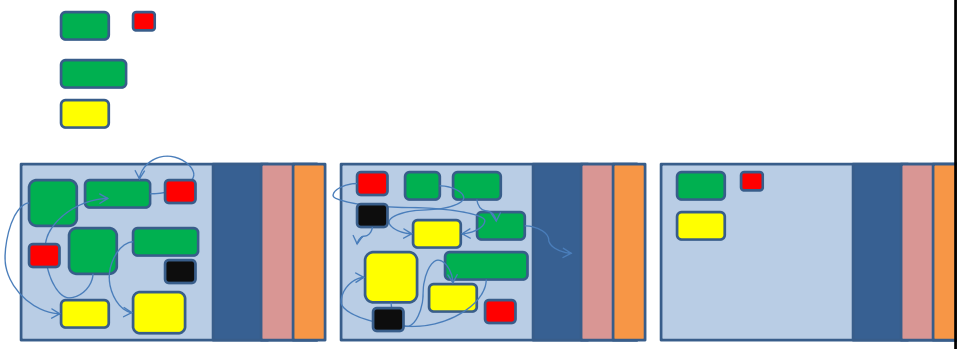
88







91



92

Constraints

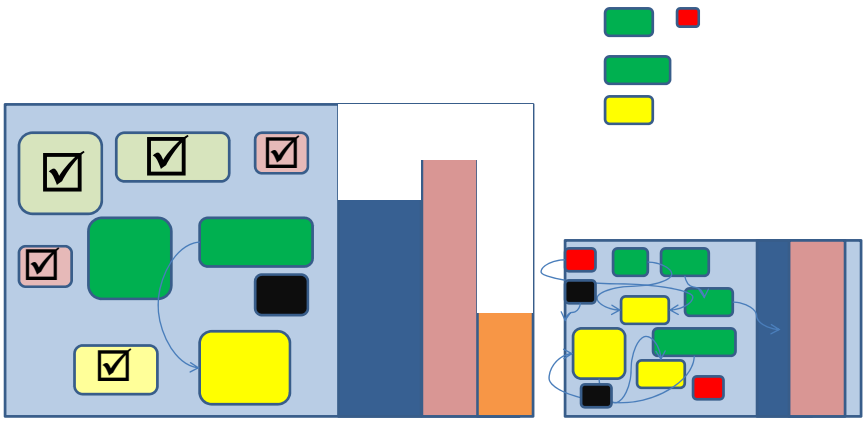
- Use rules and heuristics to do an initial plan
- Force (move) elements from one time-box to another
- Dependencies will “drag” things around
- Proceed similarly at both levels:
  - Release planning
  - Sprint planning

93

Constraints (cont.)

- Dynamically re-arrange with new information
  - Completed elements
  - Actual costs (buffer consumption)
  - New elements (in all colours)
  - New estimates
  - New dependencies
  - De-scoping
  - Additional scope
  - Loss of resource

94



95

96

Summary of Research Questions

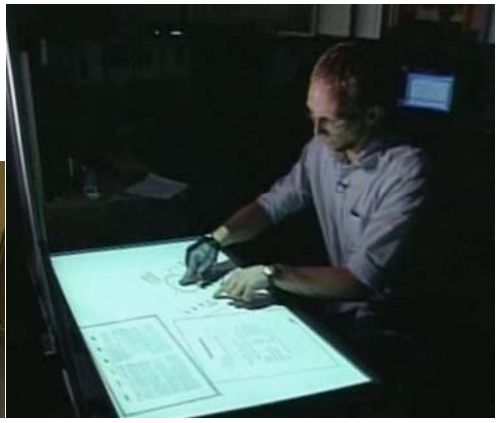
- Heuristic to allocate value to invisible features
- Heuristics to define size of buffer in a time-box
  1. For absorbing errors in estimations
  2. For defect correction
  3. For debt reduction
- Definition of the "value" of technical debt
- Impact of time on unpaid technical debt
- Visual paradigms (for tool)

Supported by Scrum Alliance

- Use of surfaces (software team room)

Support from NSERC? 97

SurfNet Research



Suggestions for project management

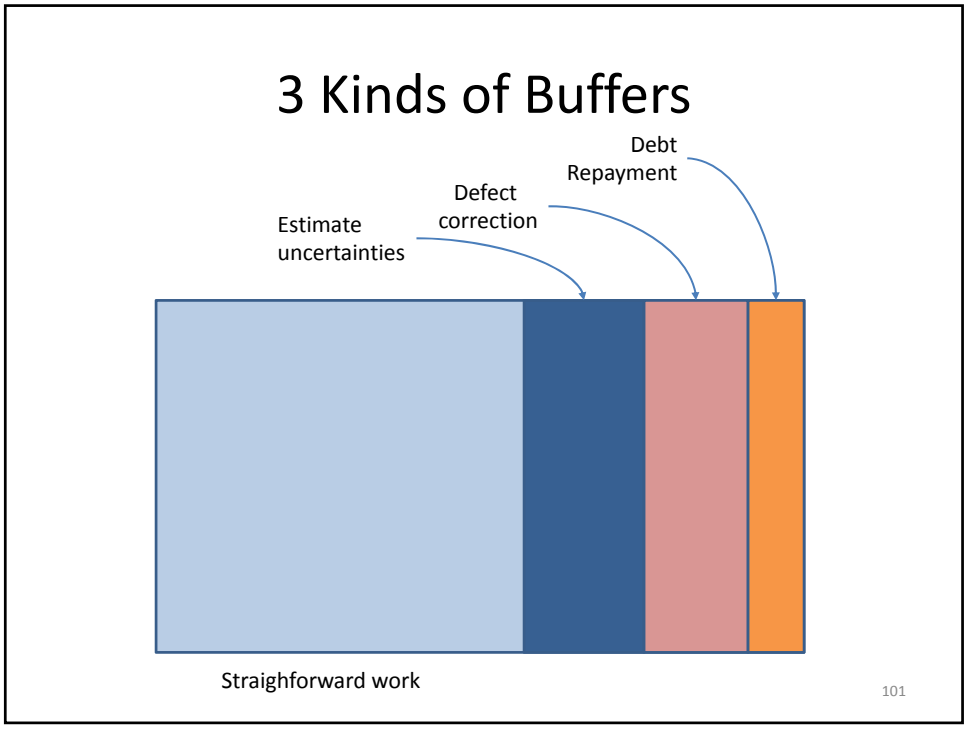
- Separate the processes for estimation of cost and value
- Avoid monetary value (points & utils)
- Identify invisible features and make them more visible to more stakeholders
- Allocate value to invisible feature
- Use nominal and worse case estimates for cost (effort); create shared buffers

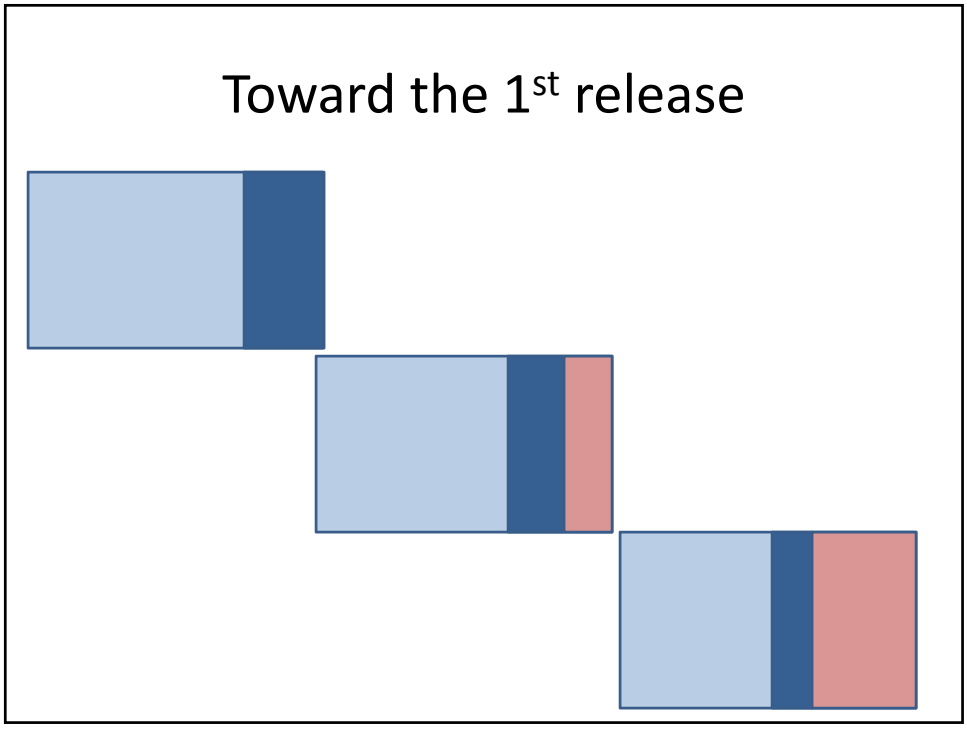
99

Suggestions (cont.)

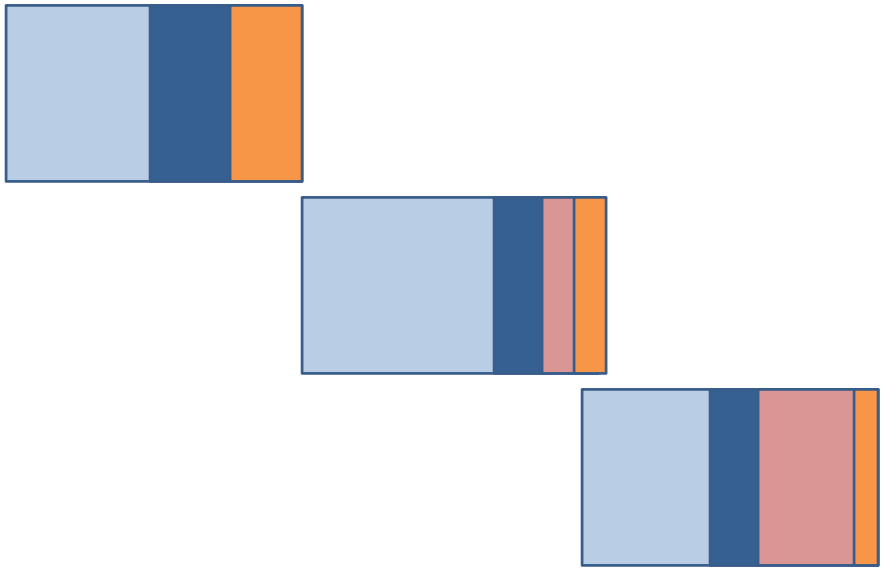
- Make technical debt visible
  - Large chunks (McConnell type 2)
- Assign some value to technical debt type 2.B and include in backlog
- Allocate a buffer in a release time-box for debt reduction for type 1 and 2.A
- Allocate a buffer in an iteration (sprint) time-box for type 1 (systematic refactorings)

100

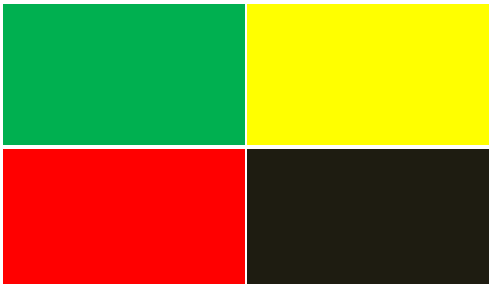




A later release



Manage all colours in your backlog!

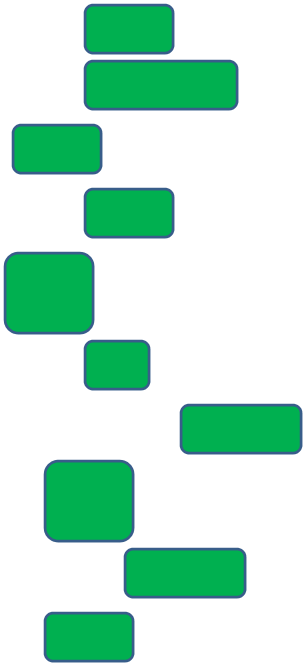


104

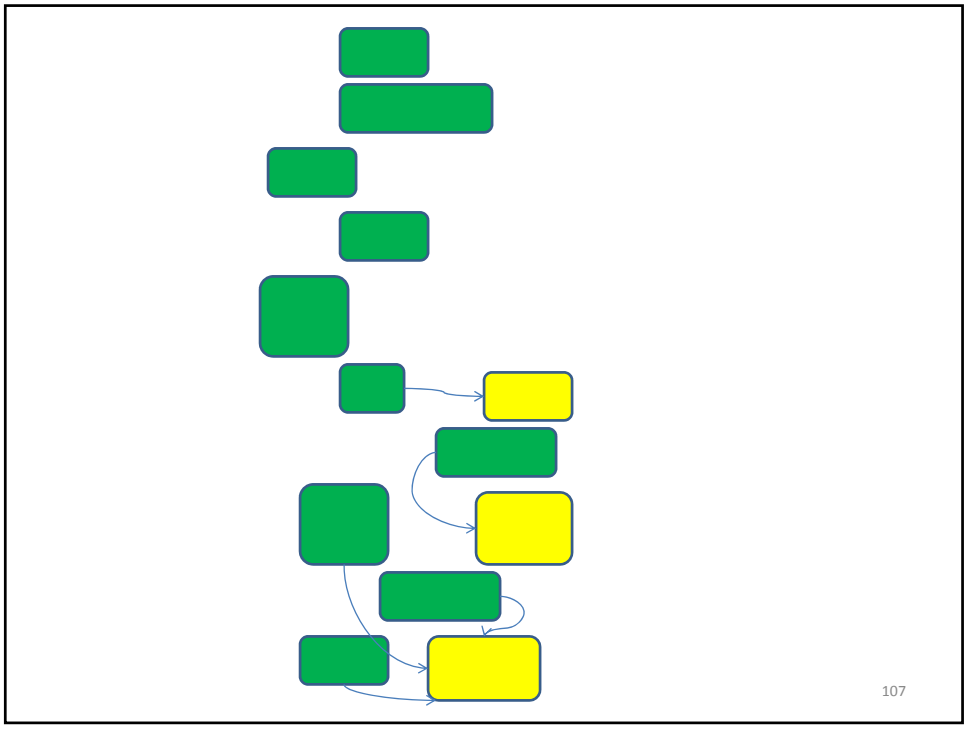
Suggestions (cont.)

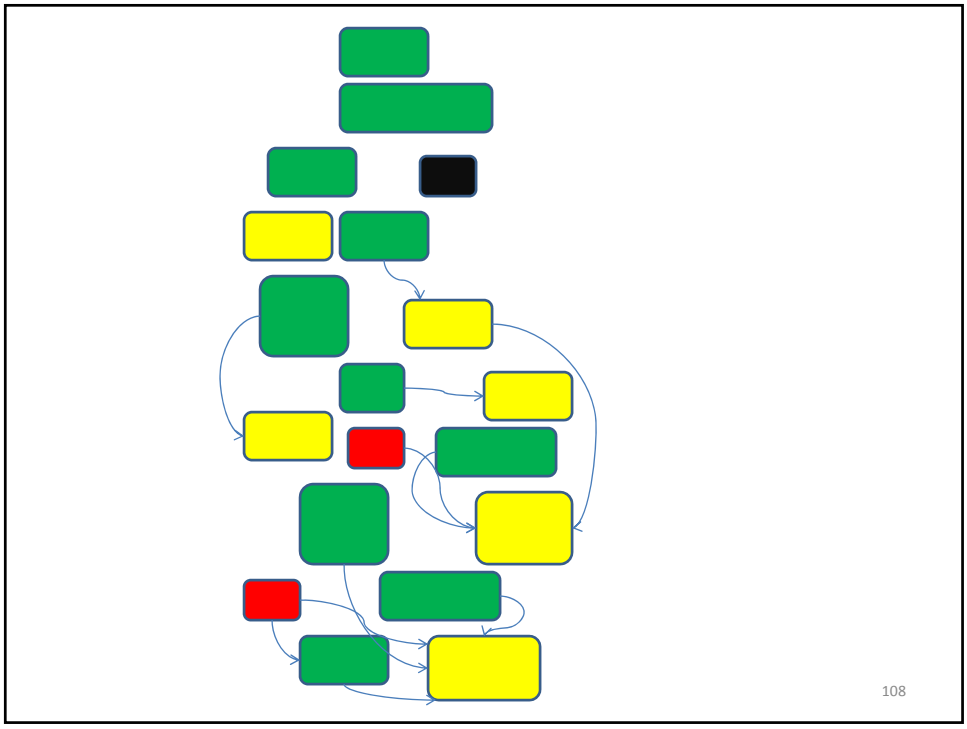
- Manage all work together, not in separate silos:
  - new development,
  - architectural or infrastructure work,
  - defect fixing and
  - debt reduction.
- Single tool…?

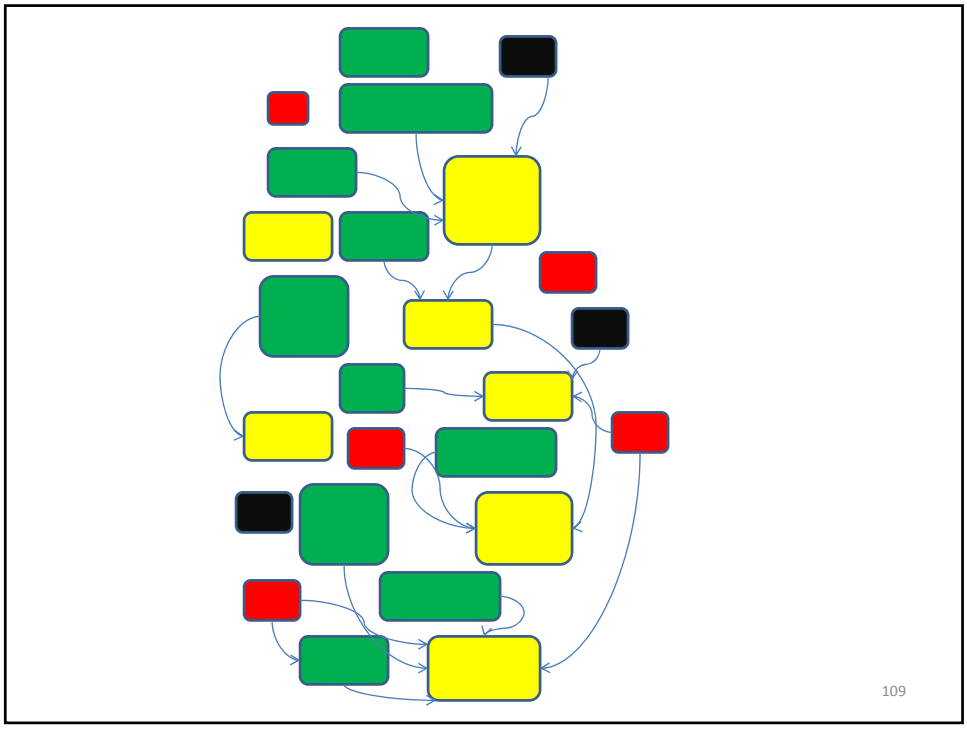
105

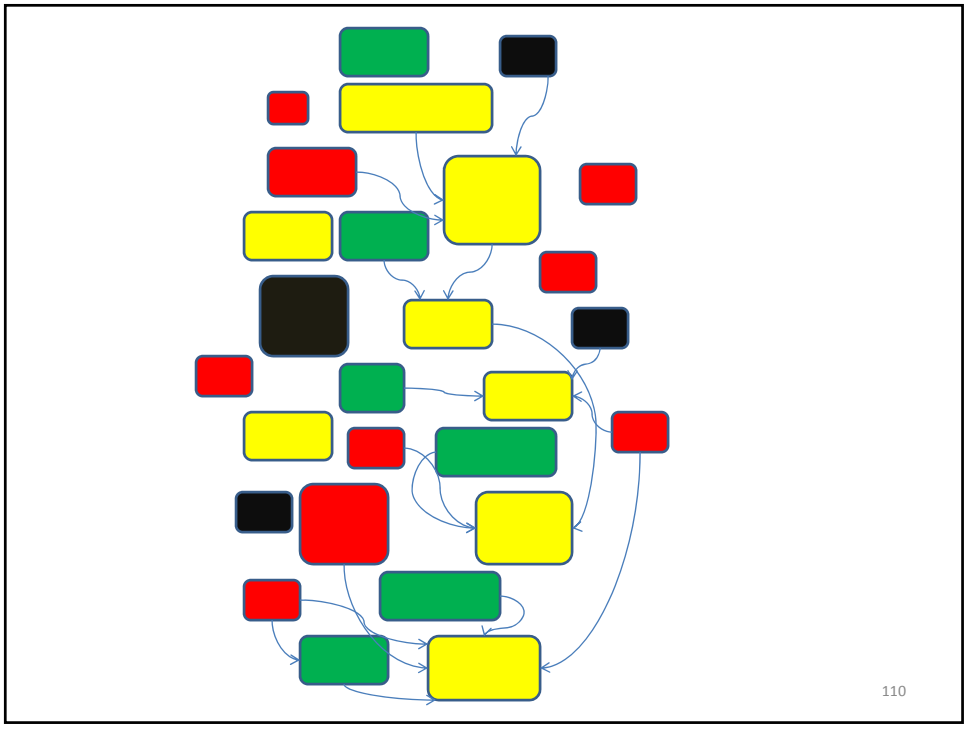


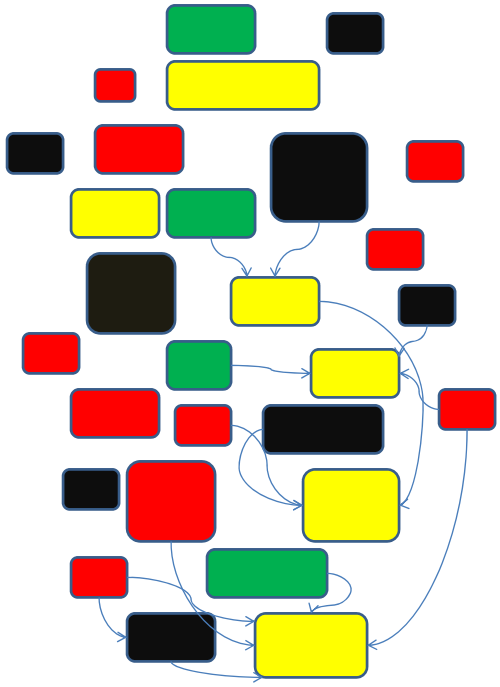
106











111

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

113

Role of Architecture

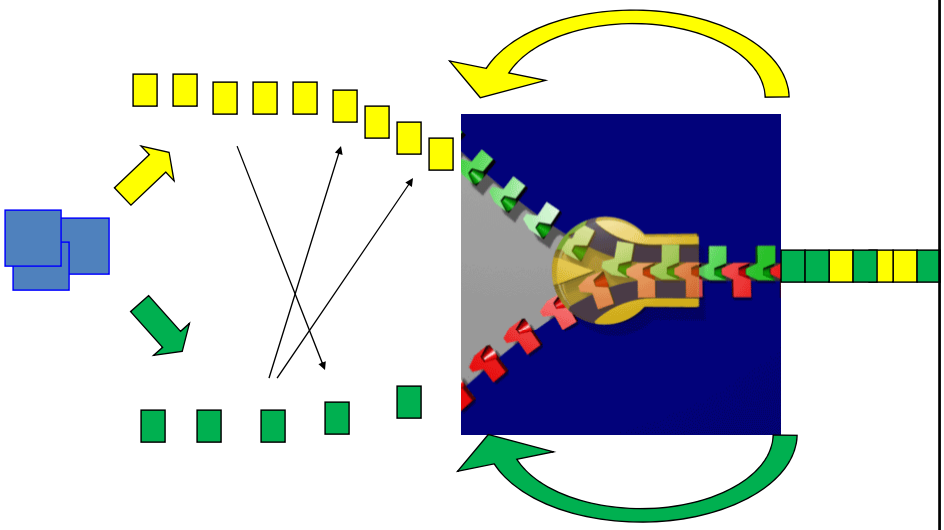
- Novel system
- Gradual emergence of architecture
- Validation of architecture with actual functionality
- Early enough to support development

Zipper model…

- Not just BUFD
- No YAGNI effect

114

Zipper: Weaving the functional and architectural work items



115

Questions?

116

References

- Agile Alliance (2001) *Manifesto for Agile Software Development.* http://agilemanifesto.org/
- 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.
- Boehm, B. and Lane, J.A. (2007) *Using the incremental commitment model to integrate system acquisition, system engineering, and software engineering,* University of Southern California, Los Angeles, September 2007.
- Boehm, B. & Ross, R. (1989) "Theory-W Software Project Management: Principles and Examples." *IEEE Transactions on Software Engineering* 15 (4) 902-916.
- Cohn, M. (2006) *Agile Estimating and Planning. Upper Saddle River, N.J.:* Prentice-Hall.
- Denne, M., & Cleland-Huang, J. (2004). *Software by Numbers: Low-Risk, High-Return Development,* Prentice Hall.

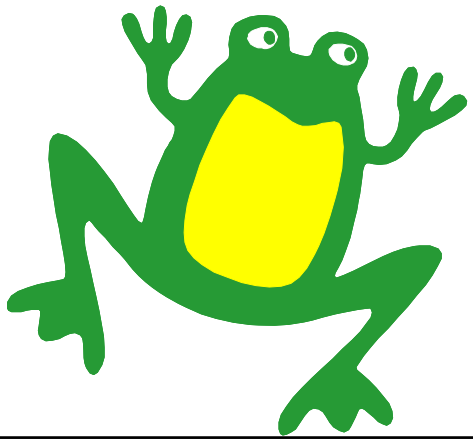
117

References (cont.)

- Denne, M., & Cleland-Huang, J. (2004). The Incremental Funding Method: Data-Driven Software Development *IEEE Software*, 21(3), 39-47.
- Karlsson, J. & Ryan, K. (1997). A Cost-Value Approach for Prioritizing Requirements, *IEEE Software,* 14 (5) 67-74.
- Kruchten, P. (2007). Voyage in the Agile Memeplex: Agility, Agilese, Agilitis, Agilology. *ACM Queue*, 5(5), 38-44.
- McConnell, S. (20087) *Notes on Technical Debt*, http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx
- Ruhe, G. and Ngo-The, A. (2004) Hybrid Intelligence in Software Release Planning. *International Journal of Hybrid Intelligent Systems,* 1, pp 99–110.
- Saaty, T. (1990). How to make a decision: The analytic hierarchy process. *European journal of operational research,* 48(1), 9-26.
- Wiegers, K. (1999). First Things First: Prioritizing Requirements. *Software Development Magazine,* 7(9), 48-53.

118

Slides at **philippe.kruchten.com/kruchten_backlog_colours.pdf**



119