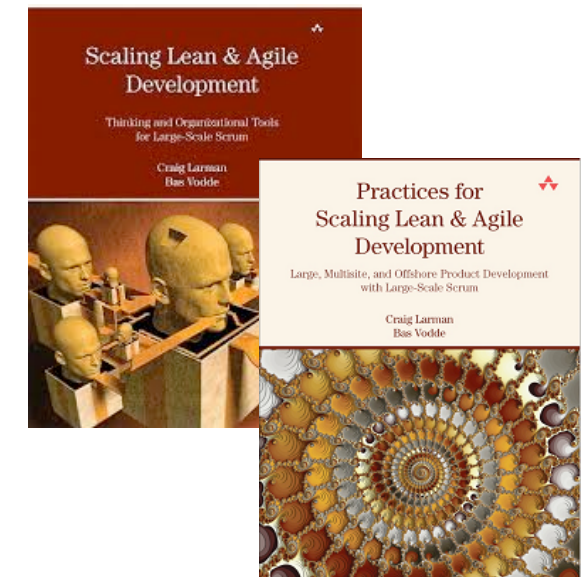
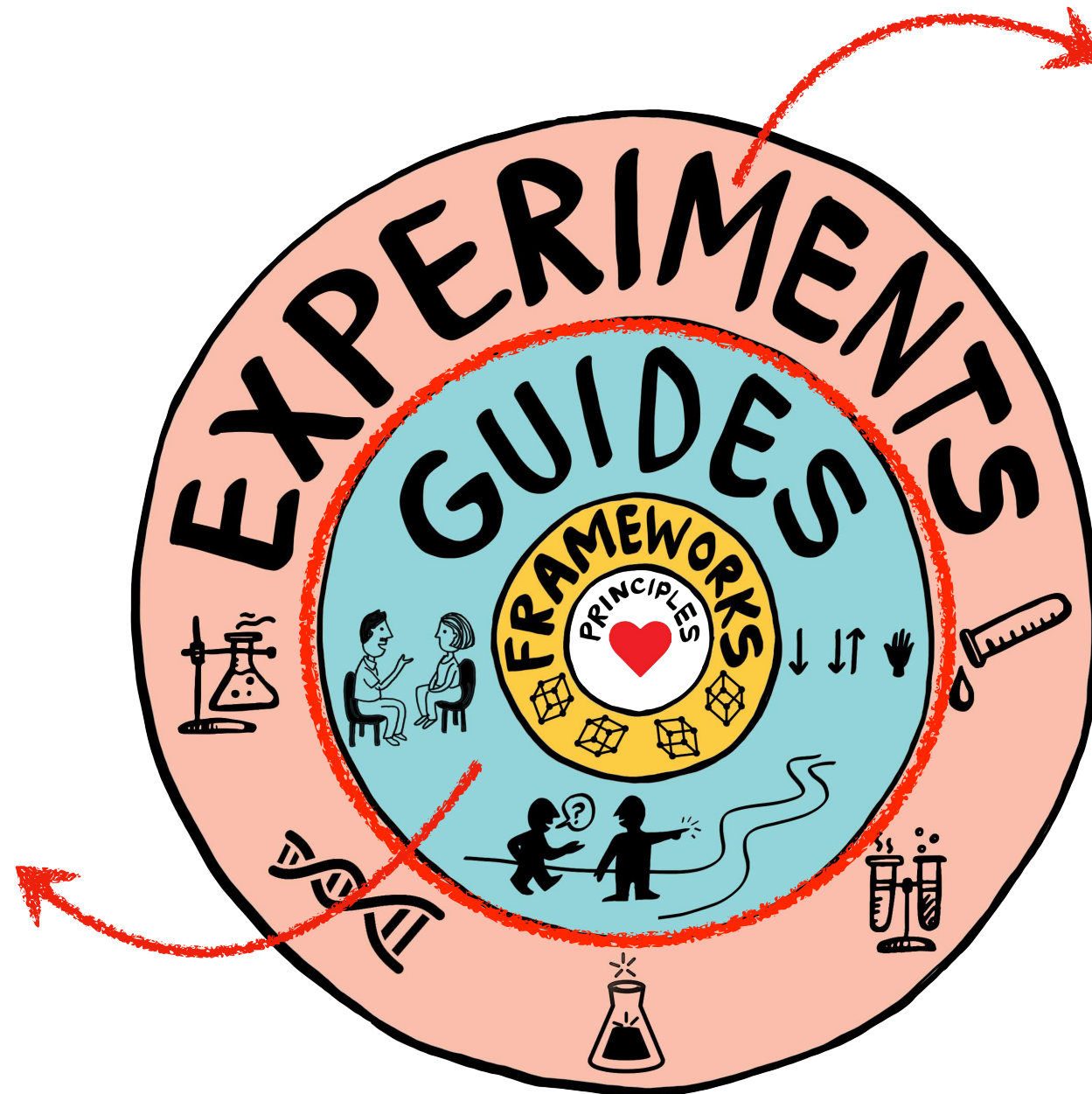


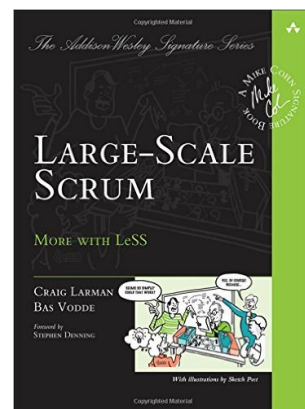
Beyond Adopting Frameworks



Lv Yi, 2024. 9



Experiment-focus



Framework-focus

1. Frameworks

LeSS Experiments

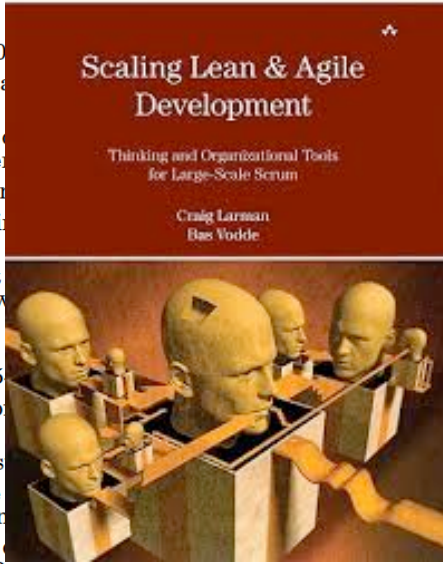
Experiments

Systems Thinking

- Try...Causal loop sketching workshop to see system dynamics 16
- Try...Sketch causal loop diagrams at whiteboards with others 16
- Try...See the positive feedback loops in your system 23
- Try...See mental models and assumptions during a causal modeling workshop 25
- Try...See root causes during causal modeling and retrospective workshops, with 5 Whys and Ishikawa diagrams 29
- Try...See and hear local optimizations; these are endemic in large product groups 32

Lean Thinking

- Avoid...Lean misconceptions 40
- Avoid...Thinking that queue management, kanban, and other tools are pillars 40
- Try...Reflect on the two pillars of lean: people and continuous improvement 40
- Try...Know system goals in lean 40
- Try...Foundation of lean thinking 48
- Try...Continuous improvement: perfection challenge, and waste flow 52
- Try...Spread knowledge rather than centralize performance to central processes 52
- Try...Study the lean meaning of waste to learn to see them 58
- Try...Improve by removing waste 58
- Try...Learn, see, and eliminate waste including handoff, overproduction, and inventory 62
- Try...Reduce the three sources of waste: inventory, overburden, NVA actions 62
- Try...Apply the 14 principles, including exceptional people, stop and fix, leveling, and pull 65
- Try...Visual management 71
- Try...Outlearn the competition 73
- Try...Long-term hands-on engineers 74
- Try...Increase the value and lower the cost of information 74
- Try...Cadence (such as timeboxing) in lean development 78
- Try...Re-use more information and knowledge through mentoring, design patterns, wikis, ... 80
- Try...Team rooms for lean development 80
- Try...Chief engineer with business acumen as chief product manager 81
- Try...Set-based concurrent engineering—several alternate designs in parallel 82



Queueing Theory

- Try...Compete on shorter cycle times 94
- Try...Use several high-level cycle-time KPIs 95
- Try...Eradicate queues by changing the system 98
- Avoid...Fake queue reduction by increased multi-tasking or utilization rates 99
- Try...Small batches of equal size 100
- Try...Visual management to see the invisible queues 111
- Try...Reduce the variability in Scrum 117
- Try...Limit size of the clear-fine subset of the Release Backlog 120

False Dichotomies

- Try...Feature teams 174

Teams

- Try...Self-organizing teams 194
- Avoid...Manager not taking responsibility for creating the conditions needed for teams to self-organize 194
- Try...Set challenging but realistic goals 195
- Try...Cross-functional teams 196
- Avoid...Single-function specialist teams 196
- Avoid...IBM 198
- Try...Long-lived teams 199
- Try...Team owns the process 200
- Try...Team manages external dependencies 202
- Try...Dedicated team members 204
- Try...Multi-skilled workers 204

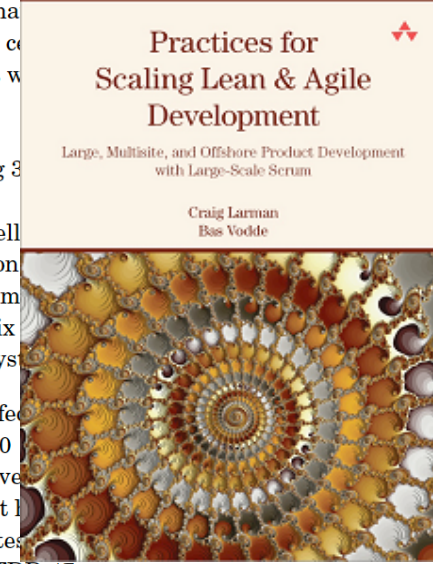
List of Experiments

Large-Scale Scrum

- Try...Large-scale Scrum FW-1 for up to ten teams 10
- Try...Large-scale Scrum FW-2 for ‘many’ teams 15

Test

- Avoid...Assuming *testing* means *testing* 24
 - Try...Challenge assumptions about testing 25
 - Avoid...Complex testing terminology 26
 - Try...Simple testing classifications 27
 - Avoid...Separating development and testing 29
 - Avoid...Test department 30
 - Avoid...Test department 32
 - Avoid...TMM, TPI, and other ‘ma
 - Avoid...ISTQB and other tester co
 - Try...Testers and programmers w
 - Try...Testers not only test 33
 - Try...Technical writer tests 34
 - Try...Educate and coach testing 3
 - Try...Community of testing 35
 - Try...Recognize project test smell
 - Avoid...Separate test automation
 - Try...Feature team as test autom
 - Try...All tests pass—stop and fix
 - Avoid...Using defect tracking sys
- iteration 39

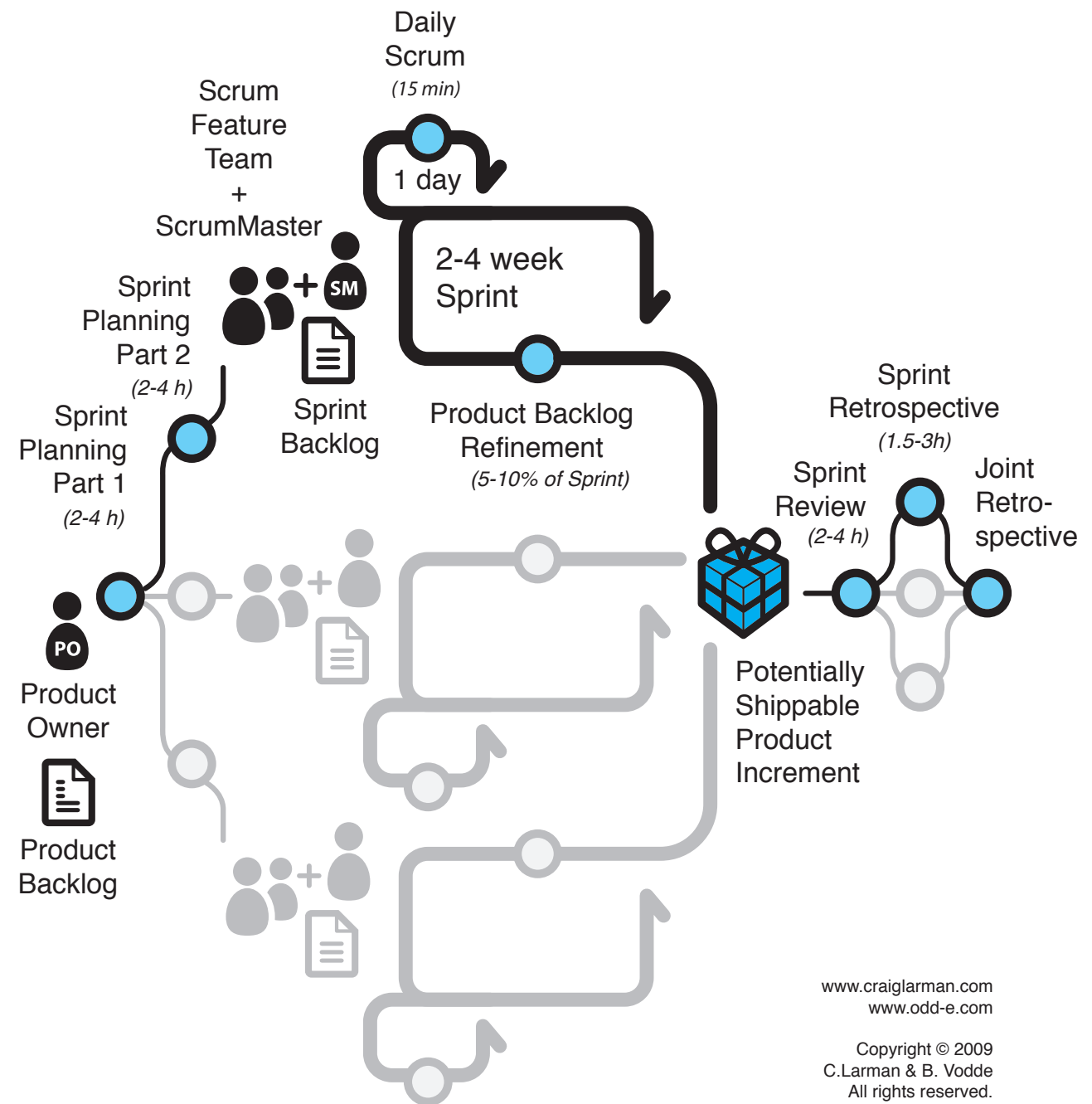


- Try...Zero tolerance on open defects 40
- Avoid...Commercial test tools 40
- Try...Acceptance test-driven development 41
- Avoid...Traditional requirement management 41
- Avoid...Thinking A-TDD is for test-driven development 41
- Avoid...Confusing TDD and A-TDD 47
- Try...A-TDD match the iteration flow 48
- Try...Discuss in workshop during Product Backlog refinement 49
- Try...Clarification over writing tests 49
- Try...Use examples 50
- Try...Product Owner writes tests 51
- Avoid...‘Optimizing’ the requirements workshop 51
- Avoid...Computers and projectors in the workshop 52
- Try...Condense workflow in business rules 52
- Try...Test the walls 52
- Try...Use table format 53
- Try...Workflow tests 54
- Try...Typical workshop agenda 54
- Try...Distill the tests 55

- Avoid...Multiple requirement descriptions 56
- Try...Use A-TDD coaches and facilitators 56
- Try...Robot Framework 57
- Try...Other A-TDD compatible tools 57
- Avoid...Conventional test tools for A-TDD 57
- Try...Wrap conventional test tools under an A-TDD tool 58
- Try...Show tests in Sprint Review 59
- Avoid...Confusing acceptance and user-acceptance test 59
- Try...Automate all tests 60
- Try...Manual tests 61
- Try...Write “A-TDD tests” for non-automatable requirements 62
- Try...Exploratory testing 62
 - Time-box exploratory test sessions 62
 - Integration System 65
 - Unit tests 65
 - Integration tests 66
 - Focus on in and between tests 66
 - Integration tests 66
 - Focus on the UI 67
 - Integration tests frequently 67
 - Integration tests 67
 - Integration tests 68
 - Integration tests functionals the same as functionals 68
 - Integration tests at area for non-functionals 70
 - Integration tests only run long-running tests 70
 - Integration tests 71
 - Integration tests 72
 - Integration tests expensive tests 72
- Try...Unit testing 72
- Try...CppUTest for C and C++ 73
- Avoid...Unit testing by a separate person 73
- Try...C++ xUnit framework for C 73
- Avoid...CUnit 73
- Try...Test-driven development 74
- Try...Use TDD coaches 74
- Try...Internal and external coaches 75
- Avoid...Write your own xUnit framework 76
- Try...Use a unit test framework in a compatible language 76
- Try...Write your own xUnit framework 76
- Try...Dual targeting 76
- Try...Run tests on the development environment 76
- Try...Run tests on the real hardware 77

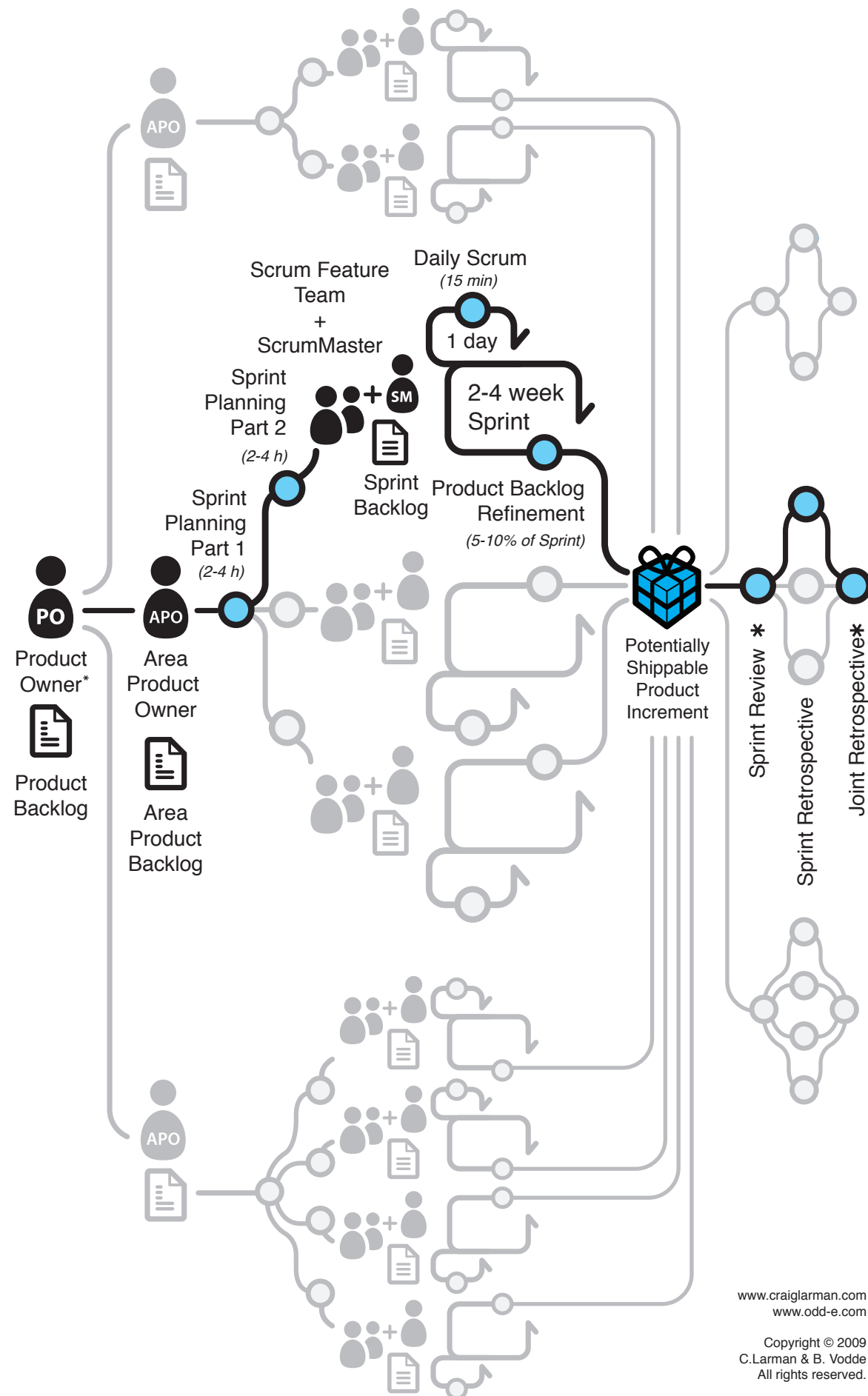
Try...

Large-Scale
Scrum **FW-1**
for up to ten
teams



Try...

Large-Scale Scrum FW-2 for 'many' teams





TRY...

Experiment rather than improve

In this light, the Try... and Avoid... ideas in this and the companion book are just experiments—and also because **systems are too complex and variable to assume prescriptive advice will work.**

LeSS Frameworks (Rules + Guides)

APPENDIX A: RULES

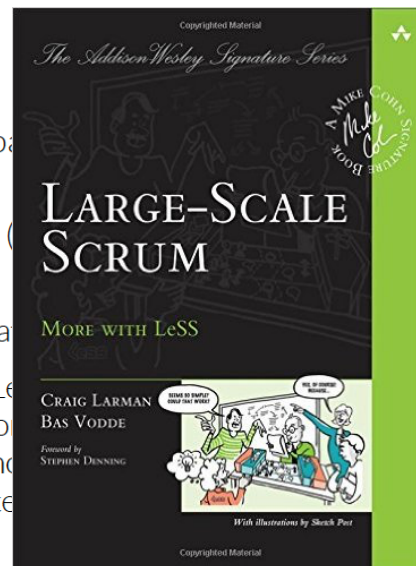
The LeSS Rules are the definition of the LeSS Framework. They are things we consider a must. Why? This is explained in the [Why LeSS](#) description on [less.works](#).

LESS FRAMEWORK RULES

The LeSS framework applies to products with 2–“8” teams.

• LeSS Structure •

- > Structure the organization using real teams as the foundational building block.
- > Each team is (1) self-managing, (2) cross-functional, (3) customer-focused, and (4) long-lived.
- > The majority of the teams are customer-focused feature teams.
- > Scrum Masters are responsible for a well-working LeSS system. Their focus is towards the Teams, Product Owner, and development practices. A Scrum Master does not manage just one team but on the overall organizational system.
- > A Scrum Master is a dedicated full-time role.
- > One Scrum Master can serve 1–3 teams.
- > In LeSS, managers are optional, but if managers do exist, their role is likely to change. Their focus shifts from managing the day-to-day product work to improving the value-delivering capability of the product development system.
- > Managers' role is to improve the product development system by practicing Go See, encouraging Stop & Fix, and “experiments over conformance.”
- > For the product group, establish the complete LeSS structure “at the start”; this is vital for a LeSS adoption.



Appendix B: Guides

More with LeSS 1

LeSS 5

Adoption 53

- Guide: Three Adoption Principles 55
- Guide: Getting Started 59
- Guide: Culture Follows Structure 64
- Guide: Job Safety but not Role Safety 66
- Guide: Organizational Perfection Vision 66
- Guide: Continuous Improvement 69
- Guide: Growing Your Adoption 71
- Guide: Evolutionary Incremental Adoption 73
- Guide: One Requirement Area at a Time 74
- Guide: Parallel Organizations 74

Organize by Customer Value 77

- Guide: Build Team-Based Organizations 79
- Guide: Understanding Feature Teams 81
- Guide: Feature-Team Adoption Maps 90
- Guide: Prefer Specialization in Customer Domain 95
- Guide: LeSS Organizational Structure 97
- Guide: Organizing Multi-Site in LeSS 100
- Guide: Requirement Areas 102
- Guide: Dynamics of Requirement Areas 105
- Guide: Transitioning to Feature Teams 106
- Guide: LeSS Huge Organization 109

Management 113

- Guide: Understand Taylor and Fayol 115
- Guide: Theory Y Management 117
- Guide: Managers Are Optional 120
- Guide: The LeSS Organization 121
- Guide: Go See 125
- Guide: Managers as Teachers and Learners 128
- Guide: Both Domain and Technical Capability 129

- Guide: LeSS Metrics with Less Targets 130
- Guide: Management Reading List 131

Scrum Masters 135

- Guide: Scrum Master Focus 137
- Guide: Five Scrum Master Tools 141
- Guide: Large-Group Facilitation 143
- Guide: Promote Learning & Multiple Skills 143
- Guide: Community Work 144
- Guide: Scrum Master Survival Guide 146
- Guide: Scrum Master Reading List 149
- Guide: Especially Pay Attention To... 150
- Guide: Avoid Requirement Area Silos 151

Product 155

- Guide: What Is Your Product? 157
- Guide: Define Your Product 162
- Guide: Expanding Product Definition 168
- Guide: Product over Project or Program 168

Product Owner 171

- Guide: Who Should be Product Owner? 173
- Guide: Start Early or Messy with a Temporary Fake Product Owner 176
- Guide: Who Are Those Users/Customers? 177
- Guide: Prioritization over Clarification 178
- Guide: Don't Do It 178
- Guide: Product Owner Helpers 179
- Guide: Five Relationships 180
- Guide: Customer Collaborations over... 187
- Guide: Ship at Least Every Sprint 189
- Guide: Don't Be Nice 189
- Guide: Let Go 190
- Guide: Don't Let Undone Work Be Your Undoing 191
- Guide: LeSS Meetings 192
- Guide: LeSS Huge Product Owner 193
- Guide: Area Product Owners 194
- Guide: PO Team Helped by Scrum Master 195



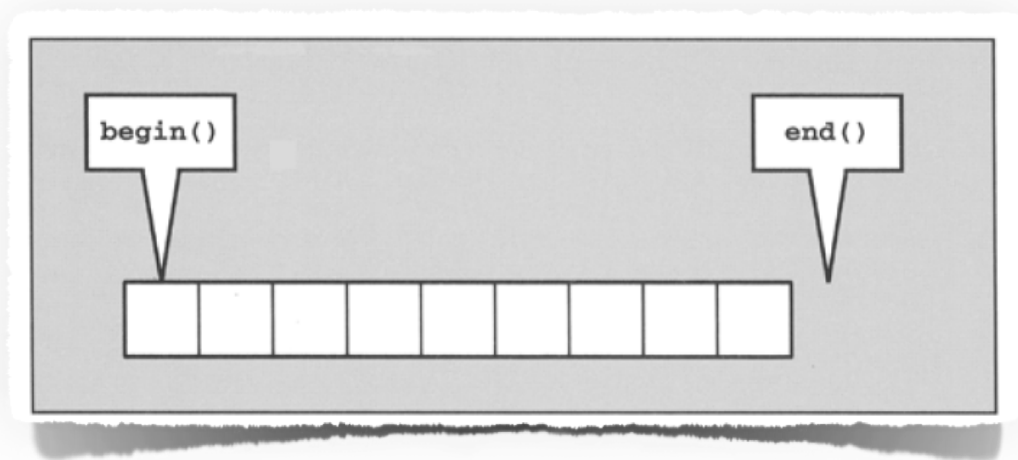
Frameworks as
starting point





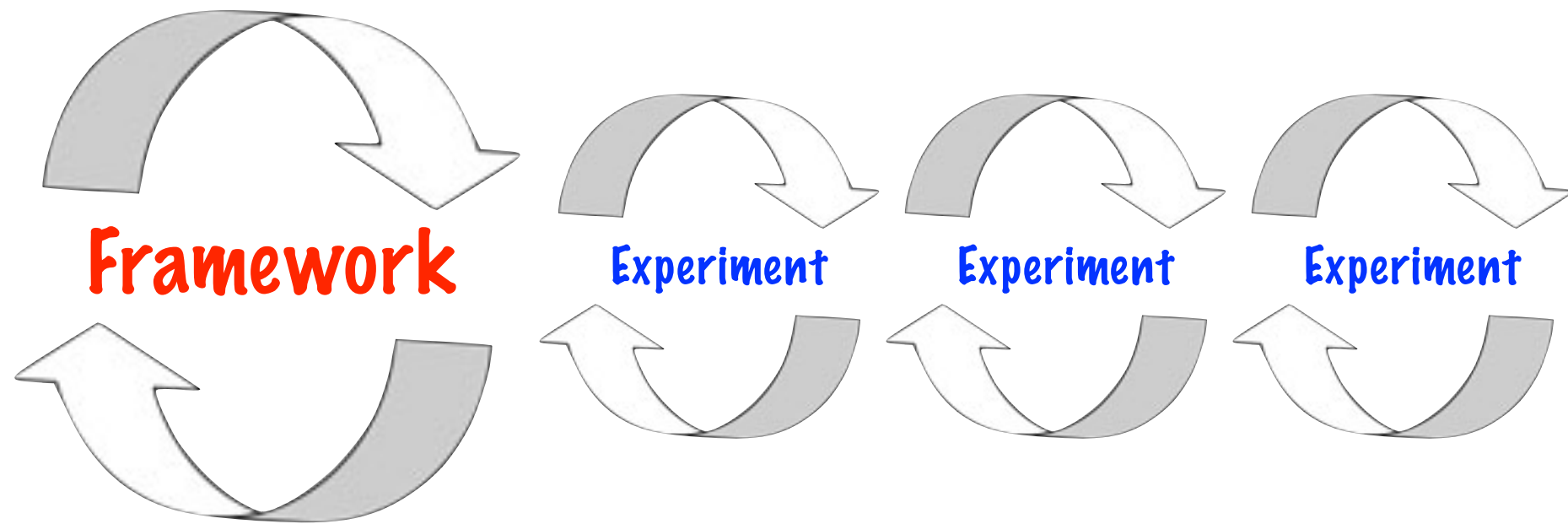
1. Improvement Continuity

Improvement is stopped
when **change project is over**



Continuous Improvement
Towards Perfection





Experiment after **Framework**



2. Improvement Ownership



1. barely sufficient

2. own, not rent



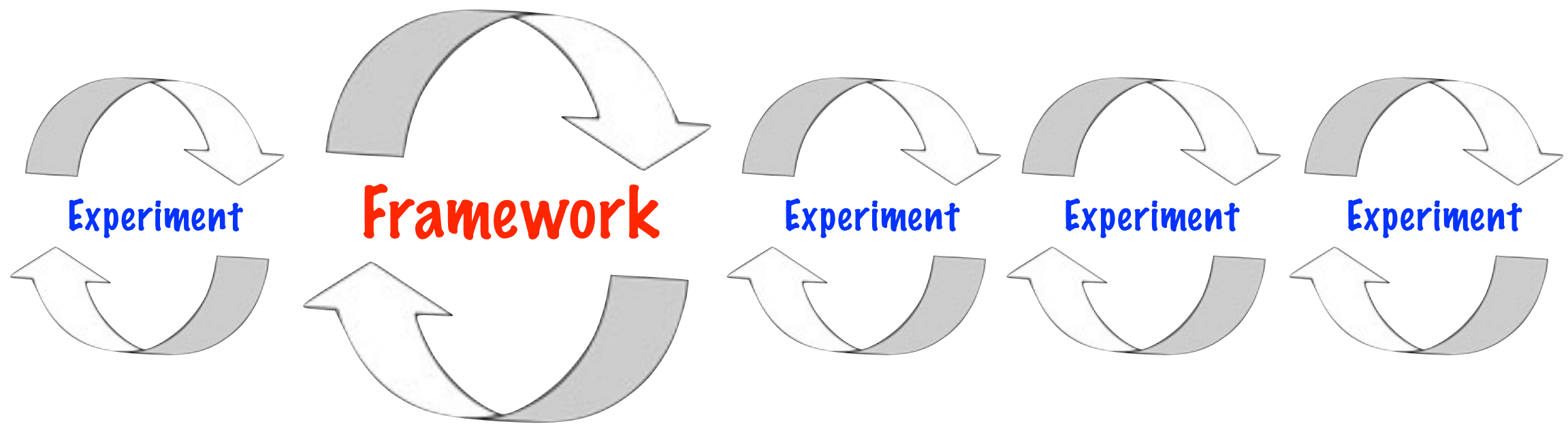
3. Improvement Capability



Big Batch change



High change capability



Experiment before Framework

2. Experiments

LeSS Experiments

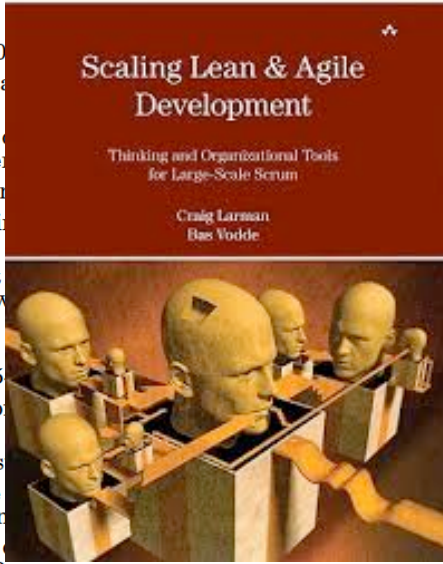
Experiments

Systems Thinking

- Try...Causal loop sketching workshop to see system dynamics 16
- Try...Sketch causal loop diagrams at whiteboards with others 16
- Try...See the positive feedback loops in your system 23
- Try...See mental models and assumptions during a causal modeling workshop 25
- Try...See root causes during causal modeling and retrospective workshops, with 5 Whys and Ishikawa diagrams 29
- Try...See and hear local optimizations; these are endemic in large product groups 32

Lean Thinking

- Avoid...Lean misconceptions 40
- Avoid...Thinking that queue management, kanban, and other tools are pillars 40
- Try...Reflect on the two pillars of lean: people and continuous improvement 40
- Try...Know system goals in lean 40
- Try...Foundation of lean thinking 48
- Try...Continuous improvement: perfection challenge, and waste flow 52
- Try...Spread knowledge rather than centralize performance to central processes 52
- Try...Study the lean meaning of waste to learn to see them 58
- Try...Improve by removing waste 58
- Try...Learn, see, and eliminate waste including handoff, overproduction, and inventory 62
- Try...Reduce the three sources of waste: inventory, overburden, NVA actions 62
- Try...Apply the 14 principles, including exceptional people, stop and fix, leveling, and pull 65
- Try...Visual management 71
- Try...Outlearn the competition 73
- Try...Long-term hands-on engineers 74
- Try...Increase the value and lower the cost of information 74
- Try...Cadence (such as timeboxing) in lean development 78
- Try...Re-use more information and knowledge through mentoring, design patterns, wikis, ... 80
- Try...Team rooms for lean development 80
- Try...Chief engineer with business acumen as chief product manager 81
- Try...Set-based concurrent engineering—several alternate designs in parallel 82



Queueing Theory

- Try...Compete on shorter cycle times 94
- Try...Use several high-level cycle-time KPIs 95
- Try...Eradicate queues by changing the system 98
- Avoid...Fake queue reduction by increased multi-tasking or utilization rates 99
- Try...Small batches of equal size 100
- Try...Visual management to see the invisible queues 111
- Try...Reduce the variability in Scrum 117
- Try...Limit size of the clear-fine subset of the Release Backlog 120

False Dichotomies

- Try...Feature teams 174

Teams

- Try...Self-organizing teams 194
- Avoid...Manager not taking responsibility for creating the conditions needed for teams to self-organize 194
- Try...Set challenging but realistic goals 195
- Try...Cross-functional teams 196
- Avoid...Single-function specialist teams 196
- Avoid...IBM 198
- Try...Long-lived teams 199
- Try...Team owns the process 200
- Try...Team manages external dependencies 202
- Try...Dedicated team members 204
- Try...Multi-skilled workers 204

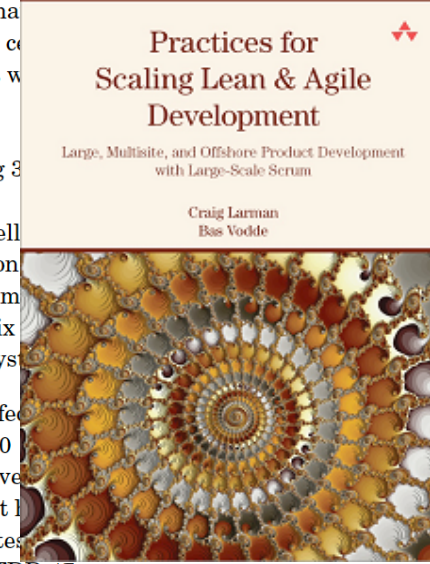
List of Experiments

Large-Scale Scrum

- Try...Large-scale Scrum FW-1 for up to ten teams 10
- Try...Large-scale Scrum FW-2 for ‘many’ teams 15

Test

- Avoid...Assuming *testing* means *testing* 24
 - Try...Challenge assumptions about testing 25
 - Avoid...Complex testing terminology 26
 - Try...Simple testing classifications 27
 - Avoid...Separating development and testing 29
 - Avoid...Test department 30
 - Avoid...Test department 32
 - Avoid...TMM, TPI, and other ‘ma
 - Avoid...ISTQB and other tester co
 - Try...Testers and programmers w
 - Try...Testers not only test 33
 - Try...Technical writer tests 34
 - Try...Educate and coach testing 3
 - Try...Community of testing 35
 - Try...Recognize project test smell
 - Avoid...Separate test automation
 - Try...Feature team as test autom
 - Try...All tests pass—stop and fix
 - Avoid...Using defect tracking sys
- iteration 39



- Try...Zero tolerance on open defects 40
- Avoid...Commercial test tools 40
- Try...Acceptance test-driven development 41
- Avoid...Traditional requirement management 41
- Avoid...Thinking A-TDD is for test-driven development 41
- Avoid...Confusing TDD and A-TDD 47
- Try...A-TDD match the iteration flow 48
- Try...Discuss in workshop during Product Backlog refinement 49
- Try...Clarification over writing tests 49
- Try...Use examples 50
- Try...Product Owner writes tests 51
- Avoid...‘Optimizing’ the requirements workshop 51
- Avoid...Computers and projectors in the workshop 52
- Try...Condense workflow in business rules 52
- Try...Test the walls 52
- Try...Use table format 53
- Try...Workflow tests 54
- Try...Typical workshop agenda 54
- Try...Distill the tests 55

- Avoid...Multiple requirement descriptions 56
- Try...Use A-TDD coaches and facilitators 56
- Try...Robot Framework 57
- Try...Other A-TDD compatible tools 57
- Avoid...Conventional test tools for A-TDD 57
- Try...Wrap conventional test tools under an A-TDD tool 58
- Try...Show tests in Sprint Review 59
- Avoid...Confusing acceptance and user-acceptance test 59
- Try...Automate all tests 60
- Try...Manual tests 61
- Try...Write “A-TDD tests” for non-automatable requirements 62
- Try...Exploratory testing 62
 - Time-box exploratory test sessions 62
 - Integration System 65
 - Unit tests 65
 - Tests 66
 - Concurrency in and between tests 66
 - Tests 66
 - Through the UI 67
 - Frequently 67
 - Priority 67
 - Priority 68
 - Functionals the same as functionals 68
 - Test area for non-functionals 70
 - Regularly run long-running tests 70
 - Regression tests 71
 - Regression tests 72
 - Regression tests 72
 - Regression tests 72
- Try...Unit testing 72
- Try...CppUTest for C and C++ 73
- Avoid...Unit testing by a separate person 73
- Try...C++ xUnit framework for C 73
- Avoid...CUnit 73
- Try...Test-driven development 74
- Try...Use TDD coaches 74
- Try...Internal and external coaches 75
- Avoid...Write your own xUnit framework 76
- Try...Use a unit test framework in a compatible language 76
- Try...Write your own xUnit framework 76
- Try...Dual targeting 76
- Try...Run tests on the development environment 76
- Try...Run tests on the real hardware 77

Which Experiments?

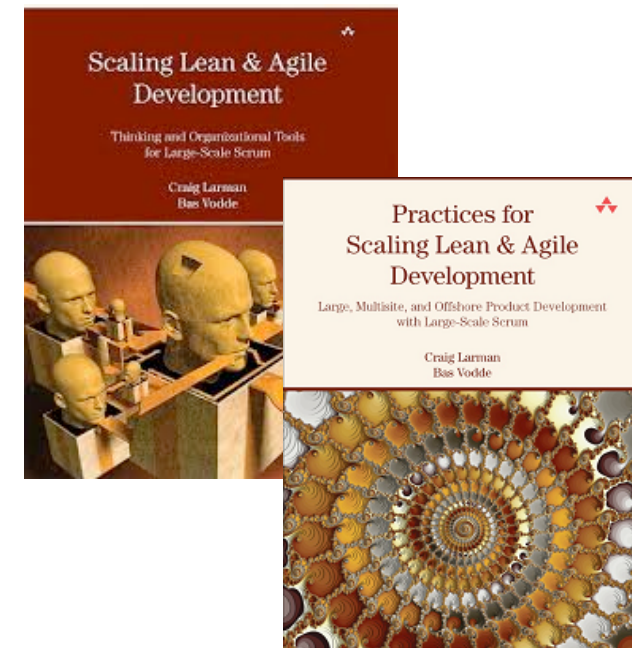


Continuous Improvement¹ inspired by LeSS experiments²

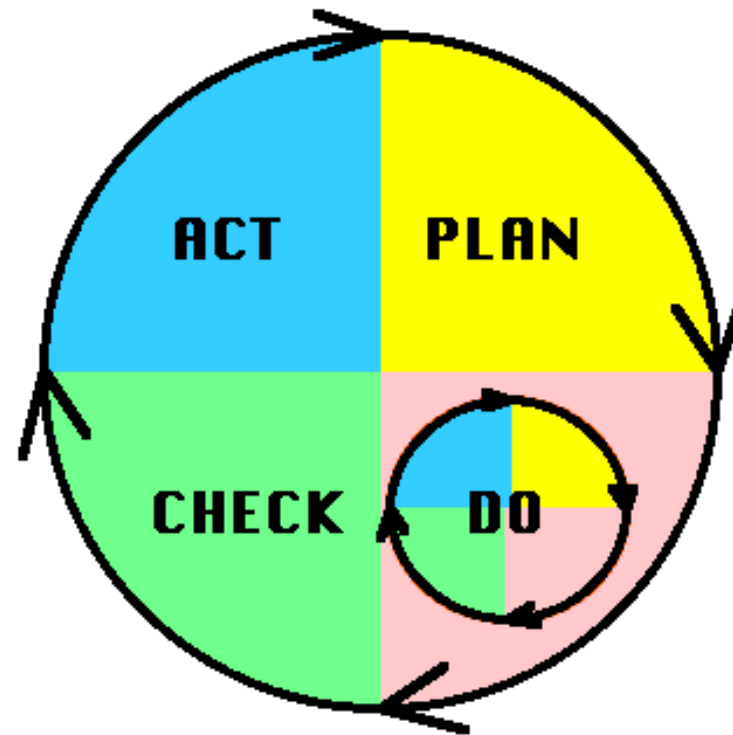
1



2



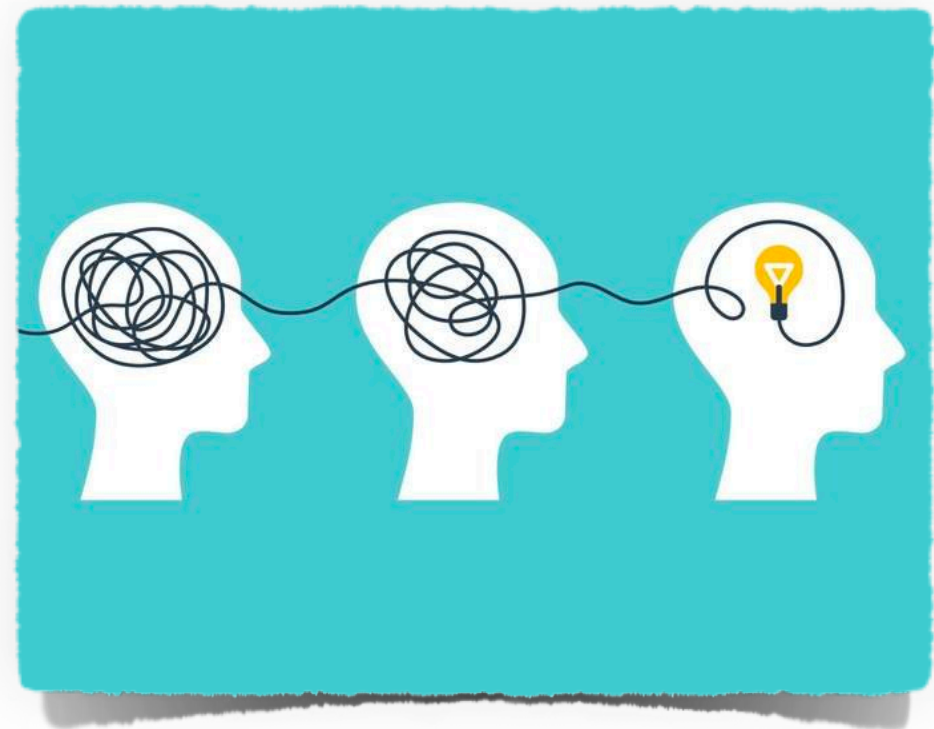
Continuous Improvement



Improvement Topic



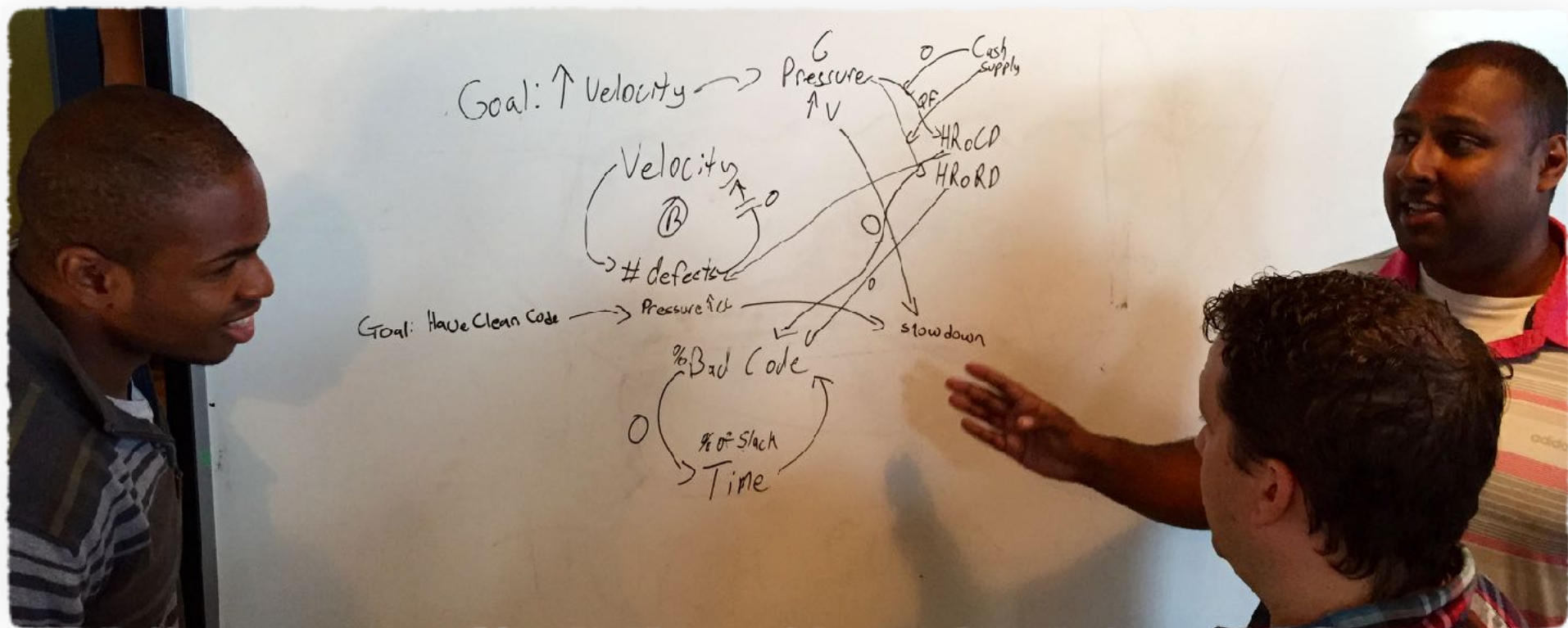
Problem



Experiment



Reasoning is at the heart of **Experiments**



Problem $\longleftrightarrow$ **System Model** $\longleftrightarrow$ Experiment

Legacy Code

Modeling 1 - improve as usual

Problems:

- Too much legacy code
- Legacy code slows us down

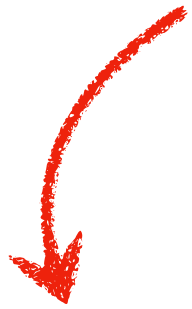
Improvement actions:

- Rewrite legacy code

Modeling 2 - browse the structure

Chapter

- How to Write New Legacy Code 334
- How to Avoid Writing New Legacy Code 335
- OK, I've Got Legacy, Now What? 343



- Avoid...Rewriting legacy code
- Try...Clean up your neighborhood
- Try...Write both high-level and unit tests
- Try...Rewrite lethal legacy code

- Avoid...Fixed content with unrealistic deadlines
- Try... Transparency and customer collaboration
- Avoid...Hiring many weak developers
- Avoid...Believing universities teach development skills
- Try...Increase organizational support for learning development skills
- Try...Support more self-study
- Avoid...Trivializing programming
- Try...Raise awareness of the negative impact of legacy code

Modeling 3 - why avoid rewriting

- Avoid. Rewriting legacy code
- Try...Clean up your neighborhood
- Try...Write both high-level and unit tests
- Try...Rewrite lethal legacy code

Key insight:

The problem is not *having* legacy code,
it is *creating* legacy code.

Modeling 4 - avoid creating new legacy

There are clear causes of legacy code:

- 1 ☐ unrealistic deadlines with fixed content
- 2 ☐ poor development skills

- 1
 - Avoid...Fixed content with unrealistic deadlines
 - Try... Transparency and customer collaboration
 - Avoid...Hiring many weak developers
 - Avoid...Believing universities teach development skills
- 2
 - Try...Increase organizational support for learning development skills
 - Try...Support more self-study
 - Avoid...Trivializing programming
 - Try...Raise awareness of the negative impact of legacy code

Modeling 5 - elaborate unrealistic deadlines

- 1 ☐ unrealistic deadlines with fixed content
- Avoid...Fixed content with unrealistic deadlines
- Try... Transparency and customer collaboration

Modeling 6 - elaborate development skills

2 ❑ poor development skills

- Avoid...Hiring many weak developers
- Avoid...Believing universities teach development skills
- Try...Increase organizational support for learning development skills
- Try...Support more self-study
- Avoid...Trivializing programming

Modeling 7 - be aware of the impact

- Try...Raise awareness of the negative impact of legacy code

Modeling 8 - clean up old legacy

- Try...Clean up your neighborhood
- Try...Write both high-level and unit tests
- Try...Rewrite lethal legacy code

Modeling 9 - refine problems and actions

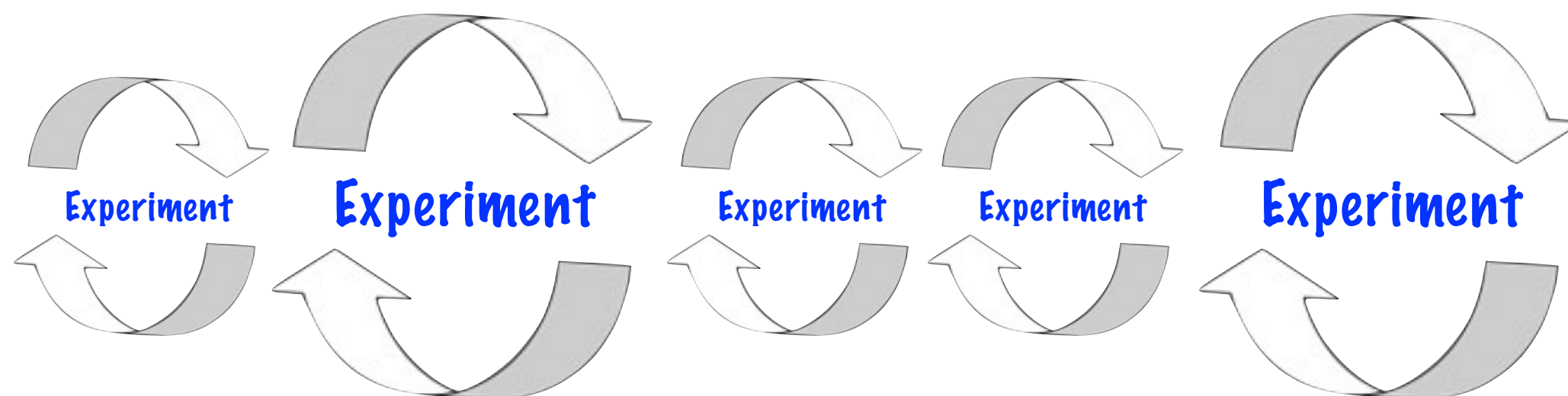
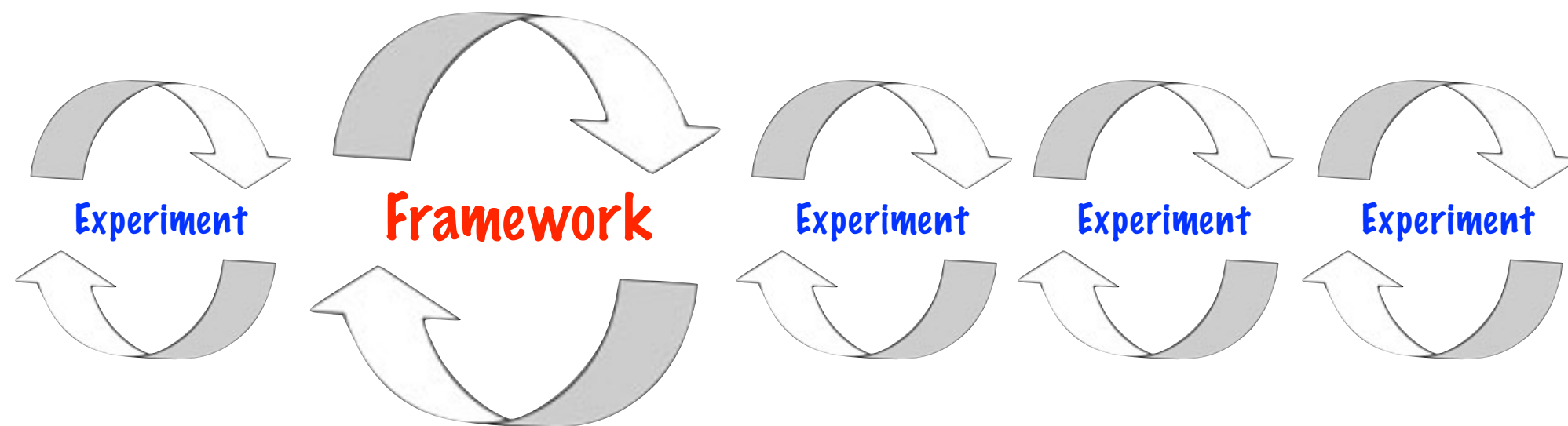
Problems:

- Too much legacy code
- Legacy code slows us down
- Unrealistic deadlines
- Poor development skills

Improvement actions:

- Rewrite legacy code (only critical part)
- Strengthen DoD
- Organize regular coding dojo

3. Conclusion



A structured way to **get inspired**



Variables

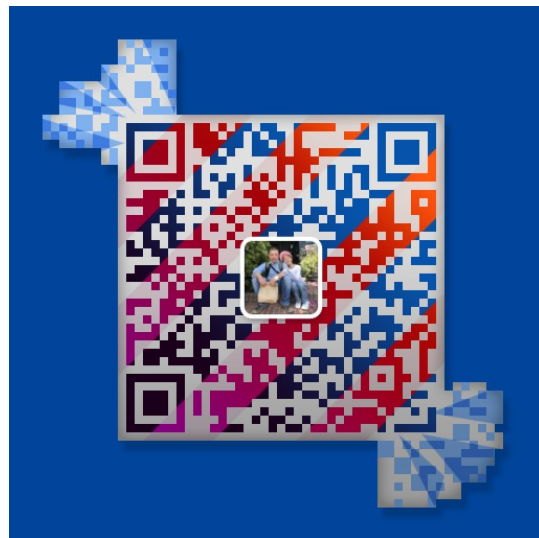
Links

Loops

References

- Beyond adopting LeSS
- Balance framework focus with experiment focus
- Experiments are at the heart of LeSS
- LeSS adoption without frameworks

Contact



Yi Lv

yi.lv@odd-e.com

<https://blog.odd-e.com/author/lv/>