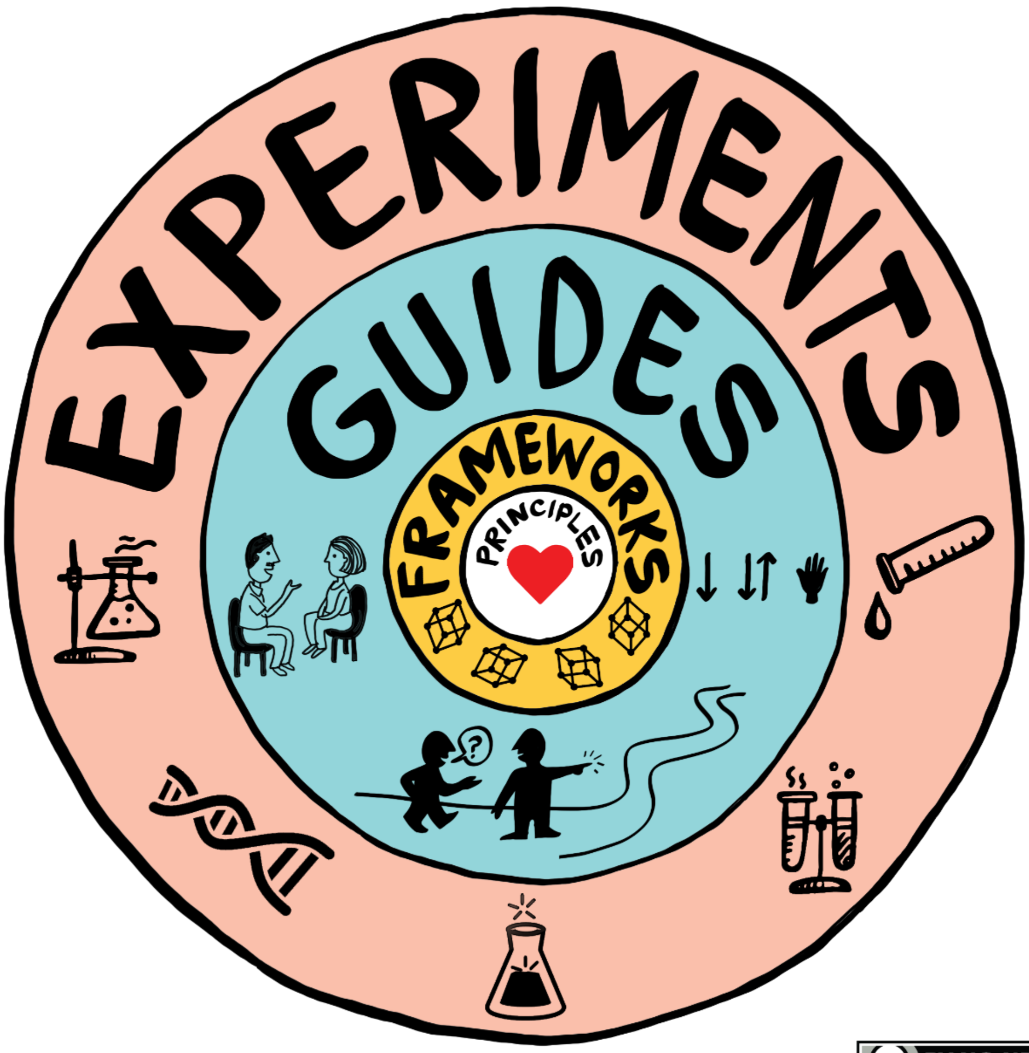
Copyright © 2014 - 2018 The LeSS Company B.V.

Large Scale Scrum (LeSS)



BY-NC-ND

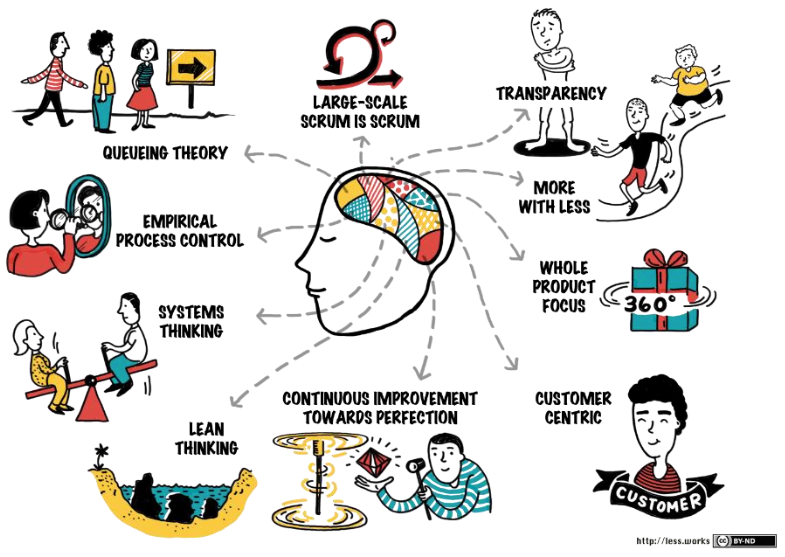
LeSS as a framework

LeSS is more than a set of principles and experiments. It also provides a framework with rules. The LeSS Rules define what is LeSS (and what isn't) and they provide a concrete framework for applying LeSS. Within the LeSS Framework, product groups can apply the experiments and discover what works best for them at a certain moment.

There are no such things as best practices. There are only practices that are good within a certain context.

Copyright © 2014 - 2018 The LeSS Company B.V.

LeSS Principles



Large-Scale Scrum is Scrum—It is not “new and improved Scrum.” LeSS is about applying the principles, elements, and purpose of Scrum in a large-scale context. Multiple-team Scrum, not multiple Scrum teams.

Empirical process control—Inspection and adaptation of the product, processes, organizational design, and practices to craft a situational appropriate organization based on Scrum, rather than following a detailed formula. And empirical process control requires and creates transparency.

Transparency—Based on tangible ‘done’ items, short cycles, working together, common definitions, and driving out fear in the workplace.

More with less—(1) In empirical process control: more learning with less defined processes. (2) In lean thinking: more value with less waste and overhead. (3) In scaling, more ownership, purpose, and joy with less roles, artifacts, and special groups.

Whole-product focus—One Product Backlog, one Product Owner, one potentially shippable product increment, one Sprint—regardless if there are 3 or 33 teams. Customers want the product, not a part.

Customer-centric—Identify value and waste in the eyes of the paying customer. Reduce the cycle time from their perspective. Increase feedback loops with real customers. Everyone understands how their work today directly relates to paying customers.

Continuous improvement towards perfection—Create and deliver a product all the time, without defects, that utterly delights customers, improves the environment, and makes lives better. Do humble and radical improvement experiments each Sprint towards that.

Systems thinking—See, understand, and optimize the whole system (not parts), and explore system dynamics. Avoid the local and sub-optimizations of focusing on the ‘efficiency’ or ‘productivity’ of individuals and individual teams. Customers care about the overall concept-to-cash cycle time and flow, not individual steps.

Lean thinking—Create an organizational system whose foundation is managers-as-teachers who apply and teach systems thinking and lean thinking, manage to improve, and who practice **Go See** at **gemba**. Add the two pillars of respect for people and continuous improvement. All towards the goal of **perfection**.

Queuing theory—Understand how systems with queues behave in the R&D domain, and apply those insights to managing queue sizes, work-in-progress limits, multitasking, work packages, and variability.

LeSS Rules (April 2018)

LeSS Framework Rules

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

LeSS Structure

- Structure the organization using real teams as the basic organizational building block.
- Each team is (1) self-managing, (2) cross-functional, (3) co-located, and (4) long-lived.
- The majority of the teams are customer-focused feature teams.
- Scrum Masters are responsible for a well-working LeSS adoption. Their focus is towards the Teams, Product Owner, organization, and development practices. A Scrum Master does not focus on 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.
- For the larger organization beyond the product group, adopt LeSS evolutionarily using Go and See to create an organization where experimentation and improvement is the norm.

LeSS Product

- There is one Product Owner and one Product Backlog for the complete shippable product.
- The Product Owner shouldn’t work alone on Product Backlog refinement; he is supported by the multiple Teams working directly with customers/users and other stakeholders.
- All prioritization goes through the Product Owner, but clarification is as much as possible directly between the Teams and customer/users and other stakeholders.
- The definition of product should be as broad and end-user/customer centric as is practical. Over time, the definition of product might expand. Broader definitions are preferred.
- One Definition of Done for the whole product common for all teams.
- Each team can have their own stronger Definition of Done by expanding the common one.
- The perfection goal is to improve the Definition of Done so that it results in a shippable product each Sprint (or even more frequently).

LeSS Sprint

- There is one product-level Sprint, not a different Sprint for each Team. Each Team starts and ends the Sprint at the same time. Each Sprint results in an integrated whole product.
- Sprint Planning consists of two parts: Sprint Planning One is common for all teams while Sprint Planning Two is usually done separately for each team. Do multi-team Sprint Planning Two in a shared space for closely related items.
- Sprint Planning One is attended by the Product Owner and Teams or Team representatives. They together tentatively select the items that each team will work on that Sprint. The Teams identify opportunities to work together and final questions are clarified.
- Each Team has their own Sprint Backlog.
- Sprint Planning Two is for Teams to decide how they will do the selected items. This usually involves design and the creation of their Sprint Backlogs.
- Each Team has their own Daily Scrum.
- Cross-team coordination is decided by the teams. Prefer decentralized and informal coordination over centralized coordination. Emphasize Just Talk and informal networks via communicate in code, cross-team meetings, component mentors, travelers, scouts, and open spaces.
- Product Backlog Refinement (PBR) is preferably done with multiple teams to increase shared learning and to exploit coordination opportunities.
- There is one product Sprint Review; it is common for all teams. Ensure that suitable stakeholders join to contribute the information needed for effective inspection and adaptation.
- Each Team has their own Sprint Retrospective.
- An Overall Retrospective is held after the Team Retrospectives to discuss cross-team and system-wide issues, and create improvement experiments. This is attended by Product Owner, Scrum Masters, Team representatives, and managers (if any).

Copyright © 2014 - 2018 The LeSS Company B.V.

LeSS Huge Framework Rules

LeSS Huge applies to products with “8+” teams. Avoid applying LeSS Huge for smaller product groups as it will result in more overhead and local optimizations.

All LeSS rules apply to LeSS Huge, unless otherwise stated. Each Requirement Area acts like the basic LeSS framework.

LeSS Huge Structure

- Customer requirements that are strongly related from a customer perspective are grouped in Requirement Areas.
- Each Team specializes in one Requirement Area. Teams stay in one area for a long time. When there is more value in other areas, teams might change Requirement Area
- Each Requirement Area has one Area Product Owner.
- Each Requirement Area has between “4-8” teams. Avoid violating this range.
- LeSS Huge adoptions, including the structural changes, are done with an evolutionary incremental approach.
- Remember each day: LeSS Huge adoptions take months or years, infinite patience, and sense of humor.

LeSS Huge Product

- One (overall) Product Owner is responsible for product-wide prioritization and deciding which teams work in which Area. He works closely with Area Product Owners.
- Area Product Owners act as Product Owners towards their teams.
- There is one Product Backlog; every item in it belongs to exactly one Requirement Area.
- There is one Area Product Backlog per Requirement Area. This backlog is conceptually a more granular view onto the one Product Backlog.

LeSS Huge Sprint

- There is one product-level Sprint, not a different Sprint for each Requirement Area. It ends in one integrated whole product.
- The Product Owner and Area Product Owners synchronize frequently. Before Sprint Planning they ensure the Teams work on the most valuable items. After the Sprint Review, they further enable product-level adaptations.

LeSS Guides

Large-Scale Scrum: More with LeSS (2015)

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

Product Backlog 197

- Guide: Don’t “Manage Dependencies” but Minimize Constraints 198
- Guide: Take a Bite 202
- Guide: Dealing with Parents 204
- Guide: Handling Special Items 207
- Guide: Tools for Large Product Backlogs 210
- Guide: More Outcome, less Output 213
- Guide: Area Backlogs 215
- Guide: Three Levels Max 222
- Guide: New Area for Giant Requirement 223
- Guide: Handling Gigantic Requirements 224

Definition of Done 229

- Guide: Creating the Definition of Done 231
- Guide: Evolve the Definition of Done 240

Product Backlog Refinement 247

- Guide: Product Backlog Refinement Types 249
- Guide: Overall PBR 251
- Guide: Multi-Team PBR 252
- Guide: Multi-Site PBR 254
- Guide: Initial PBR 255
- Guide: Splitting 260
- Guide: Scaling Estimation 269

Sprint Planning 275

- Guide: Sprint Planning One 276
- Guide: Multi-Team Sprint Planning Two 280

- Guide: No Software Tools for Sprint Backlog 281
- Guide: Product Owner Team Meeting 283

Coordination & Integration 285

- Guide: Just Talk 287
- Guide: Coordination-Friendly Environment 288
- Guide: Communicate in Code 292
- Guide: Integrate Continuously 293
- Guide: Communities 295
- Guide: Cross-Team Meetings 299
- Guide: Multi-Team Design Workshop 301
- Guide: Current-Architecture Workshop 303
- Guide: Component Mentors 304
- Guide: Open Space 305
- Guide: Travelers 306
- Guide: Scouts 307
- Guide: Maybe Don't Do Scrum of Scrums 308
- Guide: Leading Team 308
- Guide: Mix and Match Techniques 309

Review & Retrospective 313

- Guide: Adapt the Product Early and Often 315
- Guide: Review Bazaar 316
- Guide: Overall Retrospective 317
- Guide: Improve the System 320
- Guide: Multi-Area Reviews & Retrospectives 325

Copyright © 2014 - 2018 The LeSS Company B.V.

LeSS Experiments

Scaling Lean & Agile Development (2009)

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 of lean 41
- Try… Reflect on the two pillars of lean: respect for people and continuous improvement 43
- Try… Know system goals in lean thinking 46
- Try… Foundation of lean thinking manager-teachers 48
- Try… Continuous improvement with Go See, kaizen, perfection challenge, and working towards flow 52
- Try… Spread knowledge rather than force conformance to central processes 54
- Try… Study the lean meaning of value and waste; learn to see them 58
- Try… Improve by removing waste 59
- Try… Learn, see, and eliminate NVA actions including handoff, overproduction, and waiting 60
- Try… Reduce the three sources of waste: variability, 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 multitasking 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… Adjust method weight empirically in Scrum 126
- Try… Identify and avoid false dichotomies 129
- Avoid… Extreme Relativism 131
- Try… Identify misconceptions and misreads 132

Be Agile

- Try… Be agile 139
- Try… Learn and applying the four values and
- twelve agile principles for competitive advantage 141
- Try… Know and share the five Scrum values 141
- Try… Learn and applying nine agile management principles 144

Feature Teams

- Avoid… Single-function teams 155
- Avoid… Component teams 155
- 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
- Try… Team makes decisions 207
- Try… Open team conflict 208
- Avoid… Phase-based “resource allocation” 209

- Avoid… Parallel releases (a symptom of imbalanced groups and work) 209
- Avoid… Staircase branching (a symptom of imbalanced groups and work) 210
- Avoid… Projects in product development (a symptom of imbalanced groups and work) 212

Requirement Areas

- Try… One Product Owner and one Product Backlog 217
- Try… Requirements areas 218
- Try… Affinity clustering or diagram for finding requirement areas 218
- Try… Moving whole teams between areas 223
- Try… An all-at-once transition to requirement areas 224
- Avoid… Development areas 224
- Avoid… Traditional requirement management tools 226
- Avoid… Tools optimized for reporting 226

Organization

- Try… Work redesign 234
- Try… Distinguish between products and projects 236
- Avoid… Projects in product development 238
- Try… Continuous product development 238
- Try… Give projects to existing teams 239
- Avoid… Resource pools with resource management 240
- Try… Keep the organization as flat as possible 241
- Try… Make the organization slightly flatter than it can handle. 242
- Try… Invite managers to join teams to do development work 242
- Avoid… Functional units 243
- Try… Scrum teams as organizational unit 243
- Try… Organize around requirement areas 244
- Try… Keep the formal organization flexible 245
- Try… Eliminating the ‘Undone’ unit by eliminating ‘Undone’ work 245
- Try… Service and support unit 246
- Try… Internal open source for internal tools 247
- Try… Product Owner Team as organizational unit 248
- Avoid… Project Management Office 249
- Avoid… So-called Agile PMO 249
- Avoid… Fake ScrumMasters 250
- Avoid… Matrix organizations in product development 250
- Try… Self-organized team creation 251
- Try… Form self-organizing teams based on skill 252
- Try… Cultivate Communities of Practice 252
- Try… Use CoPs for functional learning 253
- Try… Merged product backlog for a set of products 256
- Try… Team works on multiple products 257
- Avoid… Stage-gate processes (if Scrum is adopted) 258
- Avoid… Especially… traditional stage-gate 260
- Avoid… Stage-gate becoming a waterfall 260
- Try… Beyond budgeting 261
- Try… Engage HR 267
- Try… Ask HR for credible research evidence 267
- Avoid… Incentives linked to performance 268
- Try… De-emphasize incentives 270
- Avoid… Putting incentives on productivity measures 271
- Try… Team incentives instead of individual incentives 272
- Try… Team-based targets without rewards 273
- Avoid… Performance appraisals 273
- Avoid… ScrumMasters do performance appraisals 275
- Try… Discuss with your team how to do appraisals 275
- Try… Fill in the forms 275
- Avoid… Limiting peoples’ perspective 276
- Avoid… Job titles 276
- Try… Create only one job title 277
- Try… Let people make their own titles; encourage funny titles 277
- Try… (if all else fails) Generic title with levels 277
- Try… Simple internal titles map to special external titles 277
- Avoid… Job descriptions 278
- Try… Simple general job descriptions 278
- Avoid… Career paths 278
- Try… Job rotation 279
- Try… Start people with job rotation 280
- Try… Hire the best 280
- Avoid… Hiring when you cannot find the best 281
- Try… Team does the hiring 281
- Try… Long and in-depth hands-on evaluation 281
- Try… Pair programming with developer candidates 282
- Try… Trial iteration 282
- Try… Lots of formal education and coaching 282
- Try… Lots of coaching 283

Large-Scale Scrum

- Try… Large-scale Scrum FW-1 for up to ten teams 291
- Try… Large-scale Scrum FW-2 for ‘many’ teams 298

Scrum Primer

- Try… Learn and do standard Scrum 308

Practices for Scaling Lean and Agile Development (2010)

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 'maturity' models 32
- Avoid… ISTQB and other tester certification 32
- Try… Testers and programmers work together 33
- Try… Testers not only test 33
- Try… Technical writer tests 34
- Try… Educate and coach testing 34
- Try… Community of testing 35
- Try… Recognize project test smells 36
- Avoid… Separate test automation team 37
- Try… Feature team as test automation team 38
- Try… All tests pass—stop and fix 38
- Avoid… Using defect tracking systems during the iteration 39
- Try… Zero tolerance on open defects 39
- Avoid… Commercial test tools 40
- Try… Acceptance test-driven development 42
- Avoid… Traditional requirement handoff 46
- Avoid… Thinking A-TDD is for testers 47
- 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 ATDD 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
- Try… Plan and time-box exploratory test sessions 64
- Try… Continuous Integration System 65
- Try… Maintainable tests 65
- Try… Refactor tests 66
- Avoid… Duplication in and between tests 66
- Try… Delete tests 66
- Avoid… Test through the UI 67
- Try… Run tests frequently 67
- Avoid… Traceability 67
- Try… Traceability 68
- Try… Treat non-functionals the same as functionals 69
- Try… Requirement area for non-functionals 70
- Try… Continuously run long-running tests 70
- Avoid… Expensive tests 71
- Try… Expensive tests 72
- Try… Automate 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… Function-to-function-pointer refactoring 78
- Try… *Learning* tests 79
- Try… Learning tests for hardware 80
- Try… Refactor tests 81
- Try… Small tests that test only one thing 82
- Avoid… Slow unit tests 83

Product Management

- Try… Exploit *business* advantages of Scrum & lean thinking 100
- Try… Understand the changes with Scrum & lean thinking 104
- Avoid… Product management negotiating a "release contract" (scope & date) with R&D 106
- Try… Product management collaborates with R&D *each iteration,* adapting release scope or date 116
- Try… Challenge traditional product-management assumptions 117
- Try… Product Manager is Product Owner 120
- Avoid… Product Manager is *not* Product Owner 120
- Avoid… Fake Product Owner 121
- Avoid… Business manager is *not* Product Owner 121
- Try… Product management owns the product 122

- Try… Product Owner owns the product 122
- Avoid… Short-term product managers or focus 123
- Try… Fake Product Owner 123
- Try… Business manager is Product Owner 124
- Avoid… Believing Product Owner is just an analyst role 124
- Avoid… Believing Product Owner *must* attend the Daily Scrum 124
- Try… Product Owner product manager focuses outward to the market and channels 124
- Avoid… Too 'inward' product management & Product Owners 124
- Avoid… Too 'outward' product management & Product Owners 125
- Avoid… Us-Them: Product Owner versus Team 125
- Avoid… "Product Owner" 126
- Try… "Product Owner" 127
- Try… Overall product manager is chief engineer 128
- Avoid… Platform group with a "shared infrastructure" backlog 128
- Try… Add and do a cross-product common goal 128
- Try… Product Owners work together to maximize company ROI 131
- Try… One and only one Product Backlog 132
- Avoid… Fake team-level "Product Backlogs" 132
- Try… Area Product Owners when many teams 133
- Try… Product Owner Team 134
- Try… Map different scaling terms 134
- Try… Better behavior over 'better' PO scaling definitions 136
- Avoid… Try… "Product Owner Team" 136
- Avoid… Too inward-focused Product Owner Team 137
- Try… Product Owner representative (supporting PO) 138
- Try… Value 139
- Avoid… Value 140
- Try… Prioritize with multiple weighted factors 141
- Try… Include total life-cycle cost of an item 142
- Avoid… Feature priority categories 143
- Avoid… False dichotomy yes/no answers to customers 145
- Try… Involve real users or customers in Sprint Review 145
- Try… Product management connects teams and customers 146
- Avoid… Product management or Product Owner between teams and users 146
- Avoid… Multi-level P-M indirection from customers to teams 146
- Try… Shift R&D language toward P-M and user language 146
- Try… Extra help for product-manager Product Owner 147
- Avoid… SMEs not talking to customers 148
- Try… Product Management inspect and adapt 148
- Try… Product management education 149
- Try… Product Managers study Scrum & attend a course 149
- Try… Product managers Go See 149
- Try… Senior product managers coach 150
- Try… Invite displaced people to join product management 150

Planning

- Try… Kickstart large-scale Scrum with *one initial* Product Backlog refinement workshop 155
- Try… Continuous product development rather than projects 157
- Try… *Initial* Product Backlog refinement workshop 158
- Try… Scaling Sprint Planning Part One 163
- Try… Simple Sprint Planning Part Two 166
- Try… Asynchronous or joint Product Backlog refinement 166
- Try… Plan bounded research or learning items 166
- Try… Plan infrastructure items by regular teams 168
- Try… Avoid… Fixing defects 169
- Try… Product-level Definition of Done 170
- Avoid… Definition of Done defined by quality group 173
- Avoid… Undone Work 173
- Avoid… Needing a Release Sprint 173
- Avoid… Needing to 'harden' 175
- Try… Include Scrum teams in a Release Sprint 175
- Try… After one Release Sprint, hand off remaining Undone Work to the Undone Unit 177
- Try… Reduce—and eventually, remove—the Undone Unit over time 178
- Try… Expand the Definition of Done 178
- Try… Expand team-level Definition of Done 179
- Try… Avoid… Early and incremental handoff of Undone Work 179
- Avoid… Try… Planning an 'agile' release train 180
- Try… Estimate with Story Points 181
- Try… Avoid… Synchronize points and range 182
- Try… Combine progress measures 183
- Try… Avoid… Estimate velocity before iteration-1 184
- Try… Adjust duration estimate with Monte Carlo simulation 184

Coordination

- Try… Avoid… Cross-department coordinator 190
- Try… Integrate all functions into the teams 191
- Try… Focus on the overall product 193
- Try… Coordinator, ambassador, and scout activities 193
- Try… Team is responsible for coordination 194
- Avoid… External-to-team coordinator 195
- Avoid… Project managers 196
- Avoid… "Fake Scrum" by renaming the project manager role 196
- Avoid… ScrumMaster coordinates 197
- Try… Facilitation (rather than coordination) by ScrumMaster 197
- Try… Focus on overall product measures 198
- Avoid… Competition between teams 198
- Try… Myriad coordination methods 199
- Try… Scrum of Scrums 200
- Try… Use different questions for the Scrum of Scrums 201
- Try… Two-part Scrum of Scrums 202
- Avoid… Scrum of Scrums being a status meeting to management 202

- Avoid… Scrum of Scrums being a ScrumMaster meeting 203
- Try… CoP for ScrumMasters 203
- Try… Rotate Scrum of Scrums representatives 203
- Avoid… Frequently rotating representatives 203
- Try… Open Space 204
- Try… Town Hall meeting 205
- Try… Joint Scrum meetings 205
- Try… Joint Sprint Review bazaar 206
- Try… Prefer decentralization solutions over centralization ones 206
- Try… Send chickens to Daily Scrums 206
- Try… Travelers 207
- Try… Communities of Practice 207
- Try… Communication CoP 208
- Try… Increase shared space 208
- Try… Break cubicles and other barriers 209
- Try… Communicate in code 211
- Try… Communicate in tests 211
- Try… Environment mapping 211
- Try… Coordination working agreements 212

Requirements & PBIs

- Try… Group items into requirement areas 215
- Try… Group items into themes 216
- Avoid… Feature screening for PBIs 216
- Try… Prune an overgrown backlog 217
- Try… Prefer cell-like splitting over treelike splitting 217
- Try… Maintain at most one ancestor—direct or indirect 220
- Try… Maintain three levels when using Area Backlogs 221
- Avoid… Maintaining more than three levels of split items 222
- Try… Use special terms for size of items 222
- Try… Defer or ignore implementation *and analysis* of sub-items 223
- Avoid… Defect items in the Product Backlog—unless few 225
- Try… Add a single placeholder PBI for all defects—when many 225
- Try… “Undone Work” and system-level NFRs as PBIs 225
- Avoid… Try… Separate “Undone Work” from the Product Backlog 226
- Try… *Genuine* research work as PBIs 227
- Try… Research items quickly lead to customer-centric PBIs 228
- Avoid… Fake research items: regular analysis, … 228
- Avoid… Giving research items to separate ‘research’ groups 228
- Try… Visual management for the Product or Release Backlog 229
- Try… Traceability with executable requirements as tests 229
- Try… Organize requirement artifacts to include… 229
- Avoid… ‘Solving’ requirement problems with a documented meta-model 232
- Avoid… A complex requirements meta-model 233
- Avoid… Describing a simple meta-model in a complex way 233
- Avoid… Separate analysis or specialist groups 234
- Avoid… Separate systems-engineering group 234
- Avoid… Separate interaction design group 235
- Avoid… Separate architecture group 235
- Avoid… Fake team members 235
- Avoid… Product Owner Team as separate analysis group 236
- Try… Write customer-centric requirements (PBIs) 236
- Avoid… Technical task ‘requirements’ (PBIs) 237
- Avoid… Technical task PBIs in team-level “Product Backlogs” 238
- Try… Ask, “Would users understand every PBI?” 238
- Try… Prefer goal-oriented over solution-oriented requirements 238
- Try… Requirements workshops 240
- Avoid… Using computers in workshops 241
- Avoid… A large queue of well-analyzed, finegrained PBIs 242
- Try… Maintain only a small queue of fine-grained PBIs 242
- Try… Requirements workshops for Product Backlog refinement 243
- Try… Specification by example—usually in tables 245
- Try… Joint requirement workshops 246
- Try… Stop refining an item once it is fully INVESTed 247
- Try… Split Product Backlog items (such as stories) 247
- Try… Ask, “What benefit from splitting in this way?” 250
- Avoid… Adopting user stories because they are ‘agile’ 266
- Avoid… Believing *writing* user stories means *user* stories 266
- Try… Apply user stories with card, conversation, confirmation 266
- Avoid… User stories good; other models bad 267
- Try… Learn *many* analysis skills: user stories, use cases, … 268
- Try… Explore requirements as automated tests 271
- Try… Prefer PBI titles in C-style user-story format—usually 271
- Avoid… Requirements management and ALM tools—for N years after agile adoption 273
- Avoid… Old-style, centralized, and hierarchical document tools 274
- Try… “Web 2.0” decentralized, networked tools 275
- Try… Baseline and version-control in your “Web 2.0” tools 275
- Avoid… Requirement information in email 276
- Try… Aggregate email and discussion threads on webpages 276
- Try… RSS feeds on requirement page changes 276
- Try… Multiple page labels for a requirement page 276

Design & Architecture

- Try… Think ‘gardening’ over ‘architecting’—Create a culture of living, growing design 282
- Try… Design workshops with agile modeling 289

- Try… Just-in-Time (JIT) modeling; vary the abstraction level 295
- Try… Design workshops each iteration 295
- Try… A couple of days to a couple of weeks of design workshops before first iteration 296
- Try… Design workshops in the team rooms 297
- Try… Joint design workshops for broader design issues 298
- Try… Technical leaders teach at workshops 299
- Try… Architects and system engineers are regular (feature) team members 300
- Avoid… System engineers and architects outside of regular feature teams 300
- Try… Serious attention to user interface (UI) skills and design 300
- Try… UI designers in regular (feature) teams 300
- Avoid… UI designers in a separate UI design group 300
- Try… Architectural analysis *before* architectural design (repeat) 301
- Try… Question all early architectural decisions as final 301
- Avoid… Conformance to outdated architectural decisions 302
- Try… Hire and strive to retain master-*programmer* 'architects' 302
- Avoid… Architecture astronauts (PowerPoint architects) 302
- Avoid… "Don't model" advice from extremists 303
- Try… Prototypes in a *different* language 304
- Try… Very early, develop a walking skeleton with tracer code 305
- Try… Incrementally build 'vertical' architectural slices of customer-centric features 305
- Try… Do customer-centric features with major architectural impact first 307
- Try… Architects clarify by programming spike solutions 308
- Avoid… Architects hand off to 'coders' 308
- Try… Tiger team conquers then divides 308
- Try… SAD workshops at end of "tiger phase" 310
- Try… Agile SAD with views & technical memos 310
- Try… Back up "human infection" with an agile SAD workshop 310
- Try… Technical leaders teach during code reviews 312
- Try… Experts participate in ongoing design workshops rather than late approval reviews 312
- Avoid… Approval reviews by experts at the end of a step 312
- Try… Design/architecture community of practice 313
- Try… Show-and-tell during workshops 313
- Try… Component guardians for architectural integrity when shared code ownership 314
- Try… Component mailing lists 314
- Try… Internal open source with teachers—for tools too 315
- Try… Configurable design for customization 315
- Avoid… Branches for customization 315
- Avoid… Create 'designs' and then send them for offshore implementation 316
- Try… Architectural and design patterns 316
- Try… Promote a shared pattern vocabulary 316
- Try… Test on the old hardware as soon as possible 317
- Try… HTML-ize and hyperlink your entire source code, daily 317
- Try… Lots of stubs, plus dependency injection 318
- Avoid… Using stubs to delay integration 319
- Try… Test-driven development for a better architecture 319
- Try… Dependency injection framework 320
- Try… Use an OS abstraction layer 320
- Try… Create a low-level hardware abstraction layer (HAL) API 320
- Try… Create a mid-level object-oriented HAL 321
- Try… Create simulation layers for hardware, etc. 321
- Try… More FPGAs and fewer ASICs 322
- Avoid… Big upfront interface design 324
- Try… Start with some weakly-typed interfaces, then strengthen 324
- Try… Simplify interface change coordination with feature teams 326
- Avoid… Freezing interfaces 327
- Try… Wrap calls to remote components with proxies or adapters 327
- Try… Start with indirect interaction between major components, then replace as needed 327

Legacy Code

- Avoid… Fixed content with unrealistic deadlines 335
- Try… Transparency and customer collaboration 337
- Avoid… Hiring many weak developers 339
- Avoid… Believing universities teach development skills 340
- Try… Increase organizational support for learning development skills 340
- Try… Support more self-study 341
- Avoid… Trivializing programming 341
- Try… Raise awareness of the negative impact of legacy code 342
- Avoid… Rewriting legacy code 343
- Try…Clean up your neighborhood 346
- Try… Write both high-level and unit tests 346
- Try… Rewrite lethal legacy code 347

Continuous Integration

- Avoid… Believing CI is a tool 352
- Avoid… Large batches of changes 355
- Avoid… Process preventing developers from checking in 357
- Avoid… Branching 358
- Try… Speed up the build 361
- Try… Add new hardware to speed up the build 362
- Try… Parallelize the build 362
- Avoid… Using ClearCase 362
- Avoid… Treating test code differently than production code 364
- Try… Multi-stage CI systems 364
- Try… A mix between feature and component CI systems 365
- Avoid… Manual promotion 365
- Try… Visual management with CI 367
- Try… Add red-green screens to your CI system 368

- Avoid… Large changes 369
- Avoid… Leaving obsolete interfaces in your code 369
- Avoid… ‘Solving’ organizational problems with technical solutions 370

Inspect & Adapt

- Avoid… Adoption with top-down management support 374
- Try… Adoption with top-down management support 375
- Try… Individuals & interactions over processes & tools 376
- Try… Job and personal safety (not role safety) 376
- Try… Patience 378
- Avoid… Adopting “do agile/lean” 378
- Avoid… Being agile/lean without agile/lean practices/tools 379
- Avoid… Agile/lean transformations or change projects 380
- Try… Agile/lean adoption *forever 381*
- Try… Impediments service rather than change management 381
- Try… Human infection 385
- Avoid… Agile/lean adoption targets or rewards 385
- Avoid… Competitive ‘improvement’ 386
- Avoid… Try… ‘Easy’ agile or lean adoption 386
- Try… Experiment rather than improve 387
- Avoid… Forcing adoption of practices 387
- Try… Encourage experiments; offer coaching 387
- Avoid… Adopting <X> because “agile didn’t work here” 388
- Avoid… IBM/Accenture/… agile adoption 388
- Avoid… Adopting agile with “agile management” tools 389
- Try… Transition from component to feature teams gradually 391
- Avoid… Waiting for the organization chart 393
- Avoid… In-line ‘ScrumMaster’ line- or project managers 393
- Try… Line manager as ScrumMaster of out-of-line team 393
- Try… Break the walls—team areas with whiteboards 394
- Try… Two-week iterations to break waterfall habits 394
- Try… One flip chart for tasks of one Product Backlog item 395
- Try… Repeating large-audience introductions 397
- Try… Open-Space Technology for early-days adoption 398
- Try… Big gatherings to share stories & experiments 398
- Try… Central coaching group 399
- Avoid… Central coaching group with formal authority 399
- Try… Concentrate the coaching on a few products 399
- Try… External agile coaches 399
- Try… Pair external agile coaches with internal ones 400
- Avoid… Advisors/consultants who are not hands-on coaches 400
- Try… Structured intensive curriculum for all teams 401
- Avoid… Internal agile/lean cookbooks 401
- Try… Joint Sprint Retrospectives 403
- Try… Joint Retrospective big improvements in Product Backlog 404
- Try… Cross-team working agreements 405
- Try… Joint Sprint Reviews 405
- Avoid… Try… Individual team-level Sprint Review 406
- Try… Spend money on improving, instead of “adding capacity” 406
- Try… Lower the waters in the lake 407
- Avoid… Rotating the ScrumMaster role quickly 408
- Try… Reduce harm of policies that cannot yet be removed 408

Multisite

- Try… Fewer sites 414
- Try… Think ‘multisite’ even when close 415
- Avoid… Believing in multisite Daily Scrum magic or that multisite forces are inconsequential 415
- Avoid… Thinking ‘distributed’ must mean ‘dispersed’ 416
- Avoid… Thinking distributed pair programming is required 416
- Try… One iteration (Sprint) for the product, not for the site 417
- Avoid… Sites organized by components or functions 417
- Try… Allocate a whole feature to a co-located feature team 418
- Avoid… Dispersed groups or ‘teams’ 419
- Try… A dispersed feature ‘team’ only if it really hurts 420
- Try… Gradual transition to co-located Scrum feature teams 421
- Try… Temporary co-location of a new dispersed team 422
- Try… Learn at existing sites, rather than add ‘expert’ sites 422
- Try… Prefer co-location of feature teams and Area Product Owner of one requirement area… but do not restrict this 423
- Try… Treat all sites as equal partners 423
- Try… Continuous integration in “one repository” across sites 424
- Try… Seeing is believing—ubiquitous cheap video technology and video culture 425
- Try… Include diverge–converge cycles in large video meetings 428
- Try… Start early multisite video meetings informally 428
- Try… Multisite planning poker (estimation poker) 429
- Try… Multisite Open Space to replace Scrum of Scrums 430
- Try… Experiment with multisite Scrum meeting formats and technologies 431 Try… Cross-pollination 432
- Try… Welcoming committees and buddies 433
- Try… Multisite communities of practice (CoP), including a communications CoP 433

- Try… Retrospectives at several levels 433
- Avoid… ScrumMaster representing the team 434
- Try… ScrumMasters acting as and encouraging matchmakers 435
- Try… Improve multisite design with Design chapter tips 435
- Try… Basic practices for multisite meetings 435
- Try… Vigilance for shared agile vocabulary and concepts 437
- Try… Cultural education 437
- Try… Vigilance about a common coding style 438
- Try… Multisite tool that records audio or video 438
- Try… Tablets for shared sketching 439
- Avoid… Commercial 'agile' tools for multisite collaboration 439
- Avoid… Commercial development tools; use free tools 440
- Try… Wikis as your share point; employ a Wiki-Gardener 440
- Avoid… ClearCase for multisite continuous integration 441

Offshore

- Try… Educate that agile offshore is not just short iterations 446
- Try… Agile guide for sales people and prospects 448
- Try… Kickoff agile workshop to educate customers 448
- Try… Remove barriers between offshore team and onshore client 450
- Try… Matchmakers rather than intermediaries 450
- Avoid… Single point of contact 450
- Try… Seeing is believing—video sessions 451
- Try… Remote Sprint Review 454
- Try… Seeing is believing—client visits team 454
- Try… Team members visit client 455
- Try… Rotating ambassadors 455
- Try… Translator on team 455
- Try… Offshore team speaks English 456
- Try… Clients participate in a Sprint Retrospective 456
- Try… Offshore group first does several iterations onshore 457
- Try… Proactively find and educate an onshore Product Owner 457
- Avoid… Believing 'yes'; ask open questions 458
- Try… Offshore requirement workshops each iteration 458
- Try… Offshore domain and vision workshop 460
- Try… Requirements documentation adaptively 'simple' 461
- Try… Frequent onshore UI prototypes 461
- Try… Semi-detailed requirements documentation for iteration 462
- Try… Detailed requirements with A-TDD 462
- Try… Wiki for all requirements 462
- Try… A-TDD for UAT 463
- Try… Manual (if you must) UAT each iteration 463
- Try… Manual pre-UAT after each feature 464
- Try… Iterative requirements onshore to offshore 465
- Try… Stable offshore Scrum teams 466
- Try… Simple titles map to special titles 467
- Try… Encouraging the teams to say 'no' 467
- Try… A ScrumMaster intent on self-organizing teams 468
- Try…Long-term agile coaching group if high attrition 468
- Try… Outside-the-site agile coaches 469
- Try… Buddy system if high attrition 469
- Avoid… Onshore management, offshore development 469
- Try… Offshoring features, not disciplines or components 470
- Try… Treating the offshore organization as internal partners 470
- Try… Dispersed feature team if us-them is a problem 472
- Avoid… Unbalanced onshore favoritism or bias 472
- Avoid… "four-year programmer" partners 473
- Try… Experts coach/review rather than dictate design 474
- Avoid… Outsourcers saying "Leave it to us, we *do agile* for you" 475
- Avoid… Outsourcers with top-heavy management 476
- Avoid… "four-year programmer" outsourcers 477
- Avoid… Outsourcers whose environment does not "walk the agile talk" 477
- Avoid… Outsourcers with analysis, coding, or testing 'factories' 478
- Avoid… Try… Large outsourcers 479
- Try… Interview outsourcer-programmers by *programming* 479
- Try… The great programmers forever 480
- Try… Improve together with your outsourcer 480
- Avoid… Believing CMMI appraisal or certification means much in creative R&D work 489
- Avoid… Believing 'agile'—or any—certification means much 493
- Avoid… Toxic CMMI consultants and appraisers 494
- Try… Alternative contract models 494
- Try… Fixed price and fixed scope with agility 495
- Avoid… Commercial tools 495

Contracts

- Try… Share these key insights with contract lawyers 500
- Try… Lawyers study agile, iterative, & systems thinking concepts 501
- Try… Appreciate a traditional lawyer's point of view 502
- Try… Debug common misunderstandings when lawyers are introduced to the third agile value 504
- Try… Lawyers study problems arising from silo mentality and lack of systems thinking 505
- Try… Lawyers study the impact of potentially deployable two-week increments on assumptions and contracts 509
- Try… Lawyers study how agility reduces risk and exposure 511
- Try… Heighten lawyer sensitivity to software project complexity by analogies to legal work 513
- Avoid… Incentives and penalties 514
- Try… Share the pain/gain 515

- Avoid… “Quality Management Plan” and “Deliverables List” 515
- Try… Collaborate early and often with lawyers 516
- Avoid…Fixed-price, fixed-scope (FPFS) contracts 531
- Try…Variable-price variable-scope progressive contracts 536
- Try…Increase flexibility in project and contract variables 538
- Try…Target-cost contracts 540
- Try…Multi-phase variable-model frameworks 543