

The EG Agile Playbook

Next Generation Agile

Exported on 11/29/2023

Table of Contents

1	1 - Introduction	26
1.1	1.1 - EG Vision and goal.....	26
1.2	1.2 - Purpose of the playbook	27
1.2.1	Content of the playbook	27
1.2.2	Target recipients of the playbook	27
1.3	1.3 - About the tooling	28
1.4	1.4 - About the processes.....	29
1.4.1	Scrum.....	30
1.4.2	KanBan	30
1.4.3	Small Scale Scrum.....	31
1.4.4	NGA Custom approaches.....	32
1.4.5	DevOps	32
2	2 - Overview	34
2.1	2.1 - The BIG picture	34
2.1.1	Process phases.....	37
2.2	2.2 - The Agile Manifesto and change of mindset.....	39
2.2.1	We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:	39
2.2.2	Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan	39
2.2.2.1	That is, while there is value in the item on the right, we value the items on the left more.	39
2.2.2.2	Individuals and interactions.....	39
2.2.2.3	Working software....	40
2.2.2.4	Customer collaboration....	40
2.2.2.5	Responding to change....	40
2.3	2.3 - The common language	40

2.3.1	The Mandatory	41
2.3.2	The Freestyle.....	41
2.3.3	CTO Common Language definition	41
2.3.3.1	Mandatory elements of the EG Common Language	41
2.3.3.2	Recommended elements of the EG Common Language	42
2.4	2.4 - Company & Scrum values	42
2.4.1	Company values.....	42
2.4.1.1	Customer Focus We have deep knowledge of our customers business and industries	43
2.4.1.2	Deliver what we promise We take responsibility and deliver mission critical solutions	43
2.4.1.3	Respect for each other We honor diverse ways of thinking.....	43
2.4.2	Scrum Values and their relation to company values	44
3	3 - The team, roles and responsibilities.....	47
3.1	3.1 - What an EG scrum team looks like and how it operates.....	47
3.1.1	The EG Scrum Team environment.....	48
3.1.2	EG Scrum Team types and detailed role distribution	49
3.1.3	The Kanban team.....	50
3.1.4	The Small Scale Scrum.....	51
3.2	3.2 - The role of Developers	51
3.2.1	Developers should strive for being:	52
3.2.2	Responsibilities and duties of a Developer (team member).....	52
3.2.3	RACI matrix for EG Scrum events and sessions	54
3.2.4	Kanban.....	54
3.2.5	RACI matrix for optional Kanban meetings and sessions.....	55
3.2.6	Small Scale Scrum	56
3.2.7	The Sprint Review, Sprint Retrospective, and Sprint Planning may be combined into a single meeting which we term the Sprint Termination, lasting approximately 90 minutes. These meetings are concise, structured (thanks to advance preparation), and must not include any unnecessary/ unrelated discussions.....	56

3.2.8 3.4.1 - The role of a Business Analyst / UX Designer (part of Developers)	58
3.2.8.1 Roles and duties of a Business Analyst / UX Designer	59
3.2.8.2 The collaboration of a Business Analyst / UX Designer with Software Engineers.....	60
3.2.8.3 The collaboration of a Business Analyst / UX Designer with the Product Owner.....	60
3.2.8.4 RACI matrix for EG Scrum events and sessions	60
3.2.9 3.4.2 - The role of SW Architect	61
3.2.9.1 SW Architects same as Developers should strive for being:	62
3.2.9.2 Responsibilities and duties of an SW Architect	63
3.2.9.3 SW Architects are a Developers from Scrum setup perspective, so they should fulfill all responsibilities related to the Developer role:	63
3.2.9.4 RACI matrix for EG Scrum events and sessions	65
3.2.9.5 Kanban.....	66
3.2.9.6 RACI matrix for optional Kanban meetings and sessions.....	66
3.2.9.7 Small Scale Scrum	67
3.2.9.8 The Sprint Review, Sprint Retrospective, and Sprint Planning may be combined into a single meeting which we term the Sprint Termination, lasting approximately 90 minutes. These meetings are concise, structured (thanks to advance preparation), and must not include any unnecessary/ unrelated discussions.....	68
3.2.10 3.4.3 - The role of QA Tester / Engineer	69
3.2.10.1 QA Testers / Engineers same as Developers should strive for being:	70
3.2.10.2 Responsibilities and duties of a QA Tester / Engineer	70
3.2.10.3 QA Testers / Engineers are a Developers from Scrum setup perspective, so they should fulfill all responsibilities related to the Developer role:	70
3.2.10.4 RACI matrix for EG Scrum events and sessions	72
3.2.10.5 Kanban.....	73
3.2.10.6 RACI matrix for optional Kanban meetings and sessions.....	73
3.2.10.7 Small Scale Scrum	75
3.2.10.8 The Sprint Review, Sprint Retrospective, and Sprint Planning may be combined into a single meeting which we term the Sprint Termination,	

lasting approximately 90 minutes. These meetings are concise, structured (thanks to advance preparation), and must not include any unnecessary/unrelated discussions.....	75
3.3 3.3 - The role of a Product Owner and Product Manager.....	77
3.3.1 Responsibilities and duties of a Product Owner.....	77
3.3.2 Responsibilities and duties of a Product Manager.....	78
3.3.3 Collaboration between the Product Owner and Product Manager.....	79
3.3.4 RACI matrix for EG Scrum events and sessions	79
3.3.5 Kanban.....	81
3.3.6 RACI matrix for optional Kanban meetings and sessions.....	81
3.3.7 Small Scale Scrum	82
3.4 3.4 - The role of a Scrum Master	83
3.4.1 EG Scrum.....	83
3.4.2 Responsibilities and duties of a Scrum Master	83
3.4.3 RACI matrix for EG Scrum events and sessions	85
3.4.4 Kanban.....	86
3.4.5 RACI matrix for optional Kanban meetings and sessions.....	86
3.4.6 Small Scale Scrum	87
3.5 3.5 - The role of an Agile Coach.....	88
3.5.1 Roles and duties of an Agile Coach.....	88
3.5.2 RACI matrix for EG Scrum events and sessions	89
3.6 3.6 - The role of a DevOps Engineer.....	90
3.6.1 Responsibilities and duties of a DevOps Engineer in EG.....	91
3.6.2 RACI matrix for EG Scrum events and sessions	92
3.7 3.7 - The role of a Manager	93
3.7.1 Responsibilities and duties of a Manager	93
3.7.2 RACI of Agile Leads responsibilities.....	94
3.7.3 RACI matrix for managers in EG Scrum events	94
3.7.4 Splitting leads responsibilities in Scrum	95
3.7.5 Anti-patterns with manager role in Scrum.....	96

3.8	3.8 - The role of a Consultant	97
3.8.1	Responsibilities and duties of a Consultant in EG	98
3.8.2	RACI matrix for EG Scrum events and sessions	98
3.9	3.9 - EG Scrum roles in various contexts.....	99
3.9.1	General EG Scrum roles RACI matrix with respect to Scrum events and activities	99
3.9.2	Collaboration and mutual responsibility matrix for EG Scrum roles	100
3.9.3	Ownership and responsibility matrix for Scrum artifacts across EG Scrum roles	106
3.9.4	Kanban.....	106
4	4 - Events and working together in sprints	107
4.1	4.1 - What an EG cadence looks like.....	107
4.1.1	Synchronized cadence in EG	107
4.1.1.1	The benefits of a synchronized cadence	109
4.1.2	EG Scrum.....	109
4.1.2.1	A common sprint naming convention	110
4.1.2.2	Sequence of events	110
4.1.3	Small Scale Scrum.....	111
4.1.4	KanBan	112
4.1.5	High-product-granularity.....	112
4.2	4.2 - The Backlog Refinement	113
4.2.1	The participants	113
4.2.2	The goal.....	113
4.2.2.1	Recommended level of Product Backlog items' readiness for upcoming sprints	114
4.2.3	The timing.....	114
4.2.4	The best practice	115
4.2.5	Small Scale Scrum.....	115
4.2.6	High granularity teams	115
4.2.7	Consultancy teams	116

4.3	4.3 - The Sprint Planning	117
4.3.1	The participants	117
4.3.2	The goal	117
4.3.2.1	Part 1 - Why is this Sprint valuable?	117
4.3.2.2	Part 2 - What can be Done this Sprint?	117
4.3.2.3	Part 3 - How will the chosen work get done?	117
4.3.2.4	What does a Product Goal look like?	118
4.3.2.5	What does a Sprint Goal look like?	118
4.3.2.6	Pre-requisites to a successful Sprint Planning	119
4.3.3	The timing	119
4.3.4	The best practice	119
4.3.5	Small Scale Scrum	120
4.4	4.4 - The Daily Scrum	120
4.4.1	The participants	120
4.4.2	The goal	120
4.4.2.1	How does it look like?	121
4.4.3	The timing	121
4.4.4	The best practice	122
4.4.5	Small Scale Scrum	122
4.4.6	Consultancy teams	122
4.5	4.5 - The Sprint Review	122
4.5.1	The participants	122
4.5.2	The goal	123
4.5.3	Preparing a good Demo	123
4.5.4	User Acceptance Testing in Agile	124
4.5.5	The timing	125
4.5.6	The best practice	125
4.5.7	Small Scale Scrum	125
4.6	4.6 - The Sprint Retrospective	126

4.6.1	The participants	126
4.6.2	The goal.....	126
4.6.2.1	5 stages of a good Sprint Retrospective	127
4.6.3	The timing.....	127
4.6.4	The best practice	128
4.7	4.7 - Daily work in a Scrum Team	128
4.7.1	Sprint priorities.....	129
4.7.2	Swarming vs. pair-programming.....	130
4.7.3	Assignment of user stories and sub-tasks.....	131
4.7.4	Bugs.....	132
4.7.5	Communication and collaboration	132
4.7.6	Daily DevOps	133
4.7.6.1	Continuous Integration.....	133
4.7.6.2	Branching models	134
4.7.6.3	Creating a branch.....	135
4.7.6.4	Committing files.....	135
4.7.6.5	Pull-requests	135
4.7.6.6	Pull requests serves many purposes:.....	136
4.7.6.7	Do not add reviewers before:	136
4.7.6.8	Reviewing a pull request.....	136
4.7.6.9	What to look for in a review?	136
4.8	4.8 - Team building	136
4.8.1	5 stages of team development	137
4.8.2	Team election.....	138
4.8.3	Team identity.....	138
4.8.4	Team personal maps	139
4.8.4.1	Create your own	139
4.8.4.2	Create one for your teammate	140
4.8.5	Team working agreement	140

4.8.6	Team expansion & modification	141
4.8.6.1	Divide & conquer	141
4.8.6.2	Shadowing.....	141
4.8.7	Team relative estimation synchronization	142
4.9	4.9 - Events for Kanban teams	143
4.9.1	Events in Kanban	143
4.9.2	Scrum with Kanban.....	147
5	5 - Managing and using artifacts	150
5.1	5.1 - What is a product vs. a project	150
5.1.1	The Product.....	151
5.1.2	The (Business) Project	151
5.2	5.2 - Refining and estimating an initial Product Backlog.....	152
5.2.1	The approach in regards to the selected agile product delivery model ...	153
5.2.2	Initial Product Backlog preparation	153
5.2.2.1	Workshop agenda	154
5.2.3	Defining the Minimum Viable Product within the Product Backlog.....	155
5.2.3.1	The MoSCoW technique	156
5.2.4	MVP in KanBan	157
5.2.5	Estimating	157
5.2.5.1	Relative estimation	157
5.2.5.2	Planning Poker	159
5.2.5.3	Common mistakes.....	160
5.3	5.3 - The Product Backlog	160
5.3.1	The Product Backlog overview.....	160
5.3.2	The Product Backlog management	162
5.3.3	The Product Backlog Items	163
5.3.3.1	User Story	163
5.3.3.2	How to write and prepare great user stories?.....	164
5.3.3.3	Epic	165

5.3.3.4	Spike	165
5.3.3.5	Sub-task.....	167
5.3.3.6	Task	168
5.3.3.7	Bugs	168
5.3.4	Managing Product Backlog - tips for Product Owner	170
5.3.5	Kanban Backlog	171
5.4	5.4 - The Sprint Backlog.....	171
5.4.1	Creating the Sprint Backlog.....	171
5.4.1.1	Multiple Scrum Teams delivering one Product.....	172
5.4.1.2	One Scrum Team delivering multiple Products	172
5.4.2	Building a plan.....	172
5.4.2.1	The Sprint Goal.....	173
5.4.3	Managing the Sprint Backlog	173
5.4.4	The DOs and DON'Ts of a Sprint Backlog	174
5.4.5	Small Scale Scrum	174
5.4.6	Kanban teams	174
5.4.7	Consultancy teams	174
5.4.8	High Product Granularity teams.....	175
5.5	5.5 - The Definition of Ready	176
5.5.1	Levels of the Definition of Ready	176
5.5.2	Working with the Definition of Ready	177
5.5.3	EG company DoR baseline	178
5.5.4	Small Scale Scrum	179
5.5.5	Kanban teams	179
5.5.6	Consultancy teams	179
5.5.7	High Product Granularity teams.....	179
5.6	5.6 - The Definition of Done.....	180
5.6.1	Levels of the Definition of Done.....	180
5.6.2	Working with the Definition of Done	181

5.6.3	EG company DoD baseline	182
5.6.4	Small Scale Scrum.....	183
5.6.5	Kanban teams	183
5.6.6	Consultancy teams	183
5.6.7	High Product Granularity teams.....	184
5.7	5.7 - Using team metrics	184
5.7.1	Velocity	184
5.7.2	Capacity.....	185
5.7.2.1	Absence planning	187
5.7.3	Effective work time	189
5.7.3.1	Slack time.....	189
5.7.4	Boards.....	189
5.7.4.1	Impediments board.....	190
5.7.4.2	Improvements board	190
5.7.5	Small Scale Scrum	190
5.7.6	Kanban.....	190
5.7.7	Consultancy teams	192
5.7.8	High Product Granularity teams.....	193
5.7.9	5.7.1 - Monte Carlo method. Statistical based forecasting for Scrum and Kanban.....	193
5.7.9.1	What is the Monte Carlo simulation	193
5.7.9.2	Flipping coin example.....	193
5.7.9.3	Standard deviation.....	196
5.7.9.4	Monte Carlo simulation for Scrum and Kanban forecasting	198
5.7.10	5.7.2 - Configuring and using Kanban Team Metrics	201
5.7.10.1	Work in progress	201
5.7.10.2	Cycle time	202
5.7.10.3	Lead Time	203
5.7.10.4	Tracking the Lead Time and the Cycle Time for the custom range of data	205

5.7.10.5 Work item age	206
5.7.10.6 Throughput.....	208
5.8 5.8 - Team page	211
5.8.1 The Team Page composition	211
5.8.1.1 The Team members, roles and responsibilities.....	211
5.8.2 Team rules, agreements, Definition of Ready, Definition of Done	212
5.8.2.1 Team working agreement	212
5.8.2.2 Definition of Ready and Definition of Done	213
5.8.3 Visualization of team metrics	213
5.8.3.1 Capacity.....	213
5.8.3.2 Velocity	213
5.8.4 Retrospective notes, sprint reports, other reports	214
5.8.4.1 Team events calendar	214
5.8.4.2 Team leave calendar.....	214
5.8.4.3 Knowledge base.....	214
5.8.5 Who should manage and update the Team page?	215
5.8.6 The page transparency.....	215
6 6 - Cross-team & cross-product collaboration	216
6.1 6.1 - When we need to reach outside the agile team.....	216
6.1.1 The first rule of scaling Agile: Don't scale	216
6.1.2 The main benefits of extending your team-product setup	218
6.1.3 The main drawbacks of extending your team-product setup	218
6.1.4 What is not recommended - try to avoid	219
6.2 6.2 - Cross-team collaboration in a single product.....	219
6.2.1 Product Owner hierarchy	219
6.2.2 Work distribution and planning synchronization	220
6.2.2.1 Additional Pre-Planning session	221
6.2.2.2 Backlog Refinement.....	221
6.2.2.3 Sprint Planning	221

6.2.3	Synchronization of daily work	222
6.2.3.1	Daily Scrum	222
6.2.3.2	Scrum of Scrums (SoS)	223
6.2.4	Sprint Review.....	223
6.2.5	Sprint Retrospective	224
6.2.6	Collaboration between teams with different setups	224
6.3	6.3 - Managing a single product backlog across multiple teams	225
6.3.1	Preparing team backlogs	226
6.3.2	Team Scrum boards	228
6.3.3	Team Kanban boards	228
6.3.4	Managing the dependencies across multiple teams.....	229
6.3.5	Dependency matrix	230
6.3.5.1	Identifying dependencies	231
6.3.5.2	The ROAM technique.....	231
6.4	6.4 - Cross-product collaboration by one or more teams.....	231
6.4.1	One team working with multiple products	232
6.4.2	Multiple teams supporting multiple products delivery.....	233
6.4.2.1	Jira overview	235
6.5	6.5 - Managing priorities, dependencies and roadmaps cross-product...	236
6.5.1	Cross-product vision, roadmaps/plans, and scope creation	236
6.5.2	Big Room Planning	237
6.5.2.1	Preparation for Big Room Planning.	237
6.5.2.2	Execution of Big Room Planning.....	238
7	7 - Jira and Confluence manuals	245
7.1	7.1 - Jira	245
7.1.1	7.1.1 - Estimating hours effort in Jira	246
7.1.1.1	Defining an estimate when creating a user story	246
7.1.1.2	Defining an estimate on an existing user story.....	248
7.1.1.3	Estimations and the Sprint (Scrum board).....	250

7.1.1.4 Planning and estimating a sprint in hours.....	253
7.1.1.5 Estimating and KanBan (board).....	254
7.1.2 7.1.2 - Planning and starting a Sprint	254
7.1.2.1 Creating a Sprint	255
7.1.2.2 Starting a Sprint	259
7.1.3 7.1.3 - Using the Active Sprint dashboard view	260
7.1.3.1 Overview	261
7.1.3.2 Columns and swimlanes	261
7.1.3.3 Transitioning and accessing issues	262
7.1.3.4 Blocking and unblocking issues	264
7.1.4 7.1.4 - Closing a sprint.....	266
7.1.4.1 Completing a sprint.....	267
7.1.5 7.1.5 - Using the KanBan board and extended backlog.....	268
7.1.5.1 The KanBan extended backlog	269
7.1.5.2 The KanBan board	270
7.1.6 7.1.6 - Jira issue types and hierarchy	272
7.1.6.1 Jira issue types	272
7.1.6.2 Jira issues hierarchy.....	276
7.1.6.3 Jira bug issue resolution types	276
7.1.7 7.1.7 - Jira issues workflow.....	279
7.1.7.1 Default Jira issues workflow (All issue types)	279
7.1.7.2 Alternative Jira issues workflow (Story, Task, Bug, Sub-Task issue types)	282
7.1.8 7.1.8 - Personalized MS Outlook rule for Jira notifications	287
7.1.8.1 Jira Notification Scheme (Default setting).....	287
7.1.8.2 Stop wathing an issue	288
7.1.8.3 Creating a filter rule in MS Outlook.....	289
7.1.9 7.1.9 - Creating custom HTML links to create Jira issues	290
7.1.9.1 Some examples of create issue URLs with prefilled parameters (fields):.....	292

7.1.10	7.1.10 - Jira JQL hints, useful examples and managing filters.....	293
7.1.10.1	Where JQL can be used?.....	296
7.1.10.2	Useful JQL examples.....	296
7.1.10.3	Use case: for nearest sprint planning - find the issues with close (2 weeks EOW) due date. You can of course customize the date range	298
7.1.10.4	Other resources.....	299
7.1.10.5	You can download and print or store below JQL Cheat Sheet that in a very synthetic and brief way is guiding on how to use JQL queries:.....	299
7.1.10.6	Exporting the JQL search results to the MS Excel Sheet for further data aggregation	299
7.1.10.7	Exporting the filter results to the Excel sheet	300
7.1.10.8	Jira filters.....	300
7.1.11	7.1.11 - Jira common boards setup for cross-product collaboration	306
7.1.11.1	Setting up common Jira board	306
7.1.11.2	Using common Jira board.....	313
7.1.12	7.1.12 - Jira Automation.....	314
7.1.12.1	Jira Software Automation - overview and benefits.....	314
7.1.12.2	Understanding the automation rules and their components	315
7.1.12.3	Getting started with Jira Automation	315
7.1.12.4	Example use cases for Jira Automation	316
7.1.12.5	Where can I practice Jira automation? Where to get support if needed? .	317
7.1.12.6	I have a proven Jira automation rule that could be used in other projects across EG - How do I share it?	317
7.1.12.7	Good practices for creating and managing Jira Automation Rules	317
7.2	7.2 - Confluence	318
7.2.1	7.2.1 - Confluence space structure.....	319
7.2.2	7.2.2 - Documenting a Sprint Retrospective.....	321
7.2.2.1	Creating a Sprint Retrospective note.....	322
7.3	7.3 - Versioning and release notes	323
7.3.1	Managing versions	324

7.3.2	Tracking and reporting releases	326
7.3.3	Adding release notes	327
7.3.4	Release burndown chart.....	328
7.3.5	Confluence reports	329
7.3.5.1	Example of Status report:.....	330
7.3.5.2	Example of Change log / release notes report:.....	332
7.3.6	7.3.1 - Documenting Release Notes	332
7.4	7.4 - Dashboard metrics description	339
7.4.1	Issue statistics widget.....	339
7.4.2	Pie chart widget	340
7.4.3	Two dimensional filter statistics widget	342
7.4.4	Created vs. resolved chart widget	343
7.5	7.5 - Logging time and transferring it to NetSuite ERP.....	344
7.5.1	Logging time in Jira	344
7.5.1.1	ERP Activity field	347
7.5.1.2	Additional Netsuite information.....	347
7.5.2	Transferring logged time to NetSuite ERP	348
7.5.2.1	Custom activities	349
7.5.3	Logging working time using Worklogs Plugin.....	350
7.5.4	Sugmitting work hours using Netsuite Chrome plugin.....	353
7.6	7.6 - User management	355
7.6.1	Attention!	356
7.6.2	Creating a new user	357
7.6.3	Changing the role of a user	359
7.6.4	Removing access for a user.....	361
7.6.5	Permission levels	361
7.6.6	Attention!	361
7.7	7.7 - Instruction to GDPR and threat assessment	364
7.7.1	Purpose	364

7.7.2	Scope	364
7.7.3	Instruction	364
7.7.3.1	Change type:.....	364
7.7.3.2	Classification of data:.....	365
7.7.3.3	Threat classification	366
7.7.3.4	Threat risk:.....	367
7.7.3.5	Risk analysis description	368
7.7.3.6	Privacy by design (Privatlivsbeskyttelse af data)	368
7.7.3.7	Access control (Adgang til data)	370
7.7.3.8	Data protection and encryption (Beskyttelse og kryptering af data).....	372
7.8	7.8 - Jira & Confluence permissions schema	373
7.8.1	EG Jira	373
7.8.2	EG Confluence.....	376
7.9	7.9 - ServiceNow integration	376
7.9.1	7.9.1 - Jira - ServiceNow integration board and issue flow.....	377
7.9.1.1	ServiceNow dedicated Jira board.....	377
7.9.1.2	"servicenow_new" label	379
7.9.1.3	Dedicated materials about ServiceNow - Jira integration.....	379
7.9.2	7.9.2 - Jira - ServiceNow integration fields mapping.....	380
7.9.2.1	Integration scope	380
7.9.2.2	Changes in Jira	380
7.9.2.3	Issue field value synchronization.....	381
7.9.3	7.9.3 - Jira - ServiceNow integration communication via comments.....	384
7.10	7.10 - [FAQ] Cloud to On-Prem migration	386
7.10.1	Jira	386
7.10.2	Confluence	395
7.11	7.11 - Jira Advanced Roadmaps	396
7.11.1	What is Jira Advanced Roadmaps?	396
7.11.2	Jira Advanced Roadmaps vs. Aha! Roadmaps	397

7.11.3	Setting and Configuring Jira Advanced Roadmaps plan.....	398
7.11.3.1	Creating Jira Plan.....	398
7.11.3.2	Configuring Jira Plan	398
7.11.3.3	Creating and managing views.....	400
7.11.3.4	Saving and managing saved views.....	403
7.11.4	Dependency management	404
7.11.4.1	Viewing the dependencies	404
7.11.4.2	Add a dependency	405
7.11.4.3	Remove dependencies	406
7.11.5	Managing releases.....	406
7.11.6	Reference materials.....	406
7.12	7.12 - EG Security default schema for GDPR data access	407
7.12.1	Introduction	407
7.12.2	Solution description.....	407
7.12.2.1	Required actions in EG Active Directory configuration.....	409
7.12.3	Summary	409
7.13	7.13 - AI Metrics for software development.....	410
7.13.1	Introduction	410
7.13.2	Metrics overview.....	410
7.13.3	Metrics measurement implementation.....	411
7.13.4	Expected measurement results	412
8	8 - Appendix.....	414
8.1	8.1 - Glossary.....	414
8.1.1	A.....	414
8.1.2	B	414
8.1.3	C	415
8.1.4	D	415
8.1.5	E	415
8.1.6	F	416

8.1.7	I	416
8.1.8	M	416
8.1.9	N.....	416
8.1.10	P	416
8.1.11	R	417
8.1.12	S	417
8.1.13	T	419
8.1.14	U.....	419
8.1.15	V	419
8.2	8.2 - FAQ	420
8.3	8.3 - Examples	427
8.3.1	8.3.1 Retrospective	427
8.3.1.1	Tools for retro:	427
8.3.1.2	Insight gathering from the last sprint:	428
8.3.1.3	Deciding what to do:	430
8.3.1.4	Closing Retrospective:.....	431
8.3.1.5	Five dysfunctions of a team - an alternative approach to retrospective ..	432
8.3.2	8.3.2 Product and Sprint Goal	433
8.3.2.1	Product Goal.....	433
8.3.2.2	Sprint Goal	434
8.3.2.3	Correlation between Product Goal and Sprint Goal	434
8.3.2.4	How to craft Sprint Goal and Product Goal:	435
8.4	8.4 - EG Agile Certification program	437
8.4.1	Table of Contents	437
8.4.2	What is the program goal?	438
8.4.3	Who is the program for?.....	438
8.4.4	What are the prerequisites to join the program?	438
8.4.5	What are the available certifications (valid for 2023)?.....	439
8.4.5.1	Professional Scrum Master I.....	439

8.4.5.2	Professional Scrum Product Owner I	439
8.4.6	How does the funding look like?.....	439
8.4.7	Course information:	439
8.4.8	Class Schedule.....	440
8.4.9	How can I sign up?.....	441
8.4.10	Course plan and useful materials	441
9	9 - Release notes.....	443
9.1	NGA(i) - version 3.0.....	444
9.2	NGA - version 1.1	445
9.3	NGA - version 1.2	446
9.4	NGA - version 1.3	447
9.5	NGA - version 1.3.1	448
9.6	NGA - version 1.4	449
9.7	NGA - version 1.5	450
9.8	NGA - version 1.6	451
9.9	NGA - version 1.7	452
9.10	NGA - version 1.8	452
9.11	NGA - version 1.9	453
9.12	NGA - version 1.10	454
9.13	NGA - version 2.0	455
9.14	NGA - version 2.1	456
9.15	NGA - version 2.2	457
9.16	NGA - version 2.3	458
9.17	NGA - version 2.4	459
9.18	NGA - version 2.4.1	460
9.19	NGA - version 2.5	461
9.20	NGA - version 2.6	462
9.21	NGA - version 2.7	463
9.22	NGA - version 2.8	464

9.23	NGA - version 2.9	465
9.24	NGA - version 2.10	466
9.25	NGA - version 2.11	467
9.26	NGA - version 3.1	468
9.27	NGA - version 3.2	469

- [1 - Introduction \(see page 26\)](#)
 - [1.1 - EG Vision and goal \(see page 26\)](#)
 - [1.2 - Purpose of the playbook \(see page 27\)](#)
 - [1.3 - About the tooling \(see page 28\)](#)
 - [1.4 - About the processes \(see page 29\)](#)
- [2 - Overview \(see page 34\)](#)
 - [2.1 - The BIG picture \(see page 34\)](#)
 - [2.2 - The Agile Manifesto and change of mindset \(see page 39\)](#)
 - [2.3 - The common language \(see page 40\)](#)
 - [2.4 - Company & Scrum values \(see page 42\)](#)
- [3 - The team, roles and responsibilities \(see page 47\)](#)
 - [3.1 - What an EG scrum team looks like and how it operates \(see page 47\)](#)
 - [3.2 - The role of Developers \(see page 51\)](#)
 - [3.4.1 - The role of a Business Analyst / UX Designer \(part of Developers\) \(see page 58\)](#)
 - [3.4.2 - The role of SW Architect \(see page 61\)](#)
 - [3.4.3 - The role of QA Tester / Engineer \(see page 69\)](#)
 - [3.3 - The role of a Product Owner and Product Manager \(see page 77\)](#)
 - [3.4 - The role of a Scrum Master \(see page 83\)](#)
 - [3.5 - The role of an Agile Coach \(see page 88\)](#)
 - [3.6 - The role of a DevOps Engineer \(see page 90\)](#)
 - [3.7 - The role of a Manager \(see page 93\)](#)
 - [3.8 - The role of a Consultant \(see page 97\)](#)
 - [3.9 - EG Scrum roles in various contexts \(see page 99\)](#)
- [4 - Events and working together in sprints \(see page 107\)](#)
 - [4.1 - What an EG cadence looks like \(see page 107\)](#)
 - [4.2 - The Backlog Refinement \(see page 113\)](#)
 - [4.3 - The Sprint Planning \(see page 117\)](#)
 - [4.4 - The Daily Scrum \(see page 120\)](#)
 - [4.5 - The Sprint Review \(see page 122\)](#)
 - [4.6 - The Sprint Retrospective \(see page 126\)](#)
 - [4.7 - Daily work in a Scrum Team \(see page 128\)](#)
 - [4.8 - Team building \(see page 136\)](#)
 - [4.9 - Events for Kanban teams \(see page 143\)](#)
- [5 - Managing and using artifacts \(see page 150\)](#)
 - [5.1 - What is a product vs. a project \(see page 150\)](#)
 - [5.2 - Refining and estimating an initial Product Backlog \(see page 152\)](#)
 - [5.3 - The Product Backlog \(see page 160\)](#)
 - [5.4 - The Sprint Backlog \(see page 171\)](#)

- 5.5 - The Definition of Ready (see page 176)
- 5.6 - The Definition of Done (see page 180)
- 5.7 - Using team metrics (see page 184)
 - 5.7.1 - Monte Carlo method. Statistical based forecasting for Scrum and Kanban (see page 193)
 - 5.7.2 - Configuring and using Kanban Team Metrics (see page 201)
- 5.8 - Team page (see page 211)
- 6 - Cross-team & cross-product collaboration (see page 216)
 - 6.1 - When we need to reach outside the agile team (see page 216)
 - 6.2 - Cross-team collaboration in a single product (see page 219)
 - 6.3 - Managing a single product backlog across multiple teams (see page 225)
 - 6.4 - Cross-product collaboration by one or more teams (see page 231)
 - 6.5 - Managing priorities, dependencies and roadmaps cross-product (see page 236)
- 7 - Jira and Confluence manuals (see page 245)
 - 7.1 - Jira (see page 245)
 - 7.1.1 - Estimating hours effort in Jira (see page 246)
 - 7.1.2 - Planning and starting a Sprint (see page 254)
 - 7.1.3 - Using the Active Sprint dashboard view (see page 260)
 - 7.1.4 - Closing a sprint (see page 266)
 - 7.1.5 - Using the KanBan board and extended backlog (see page 268)
 - 7.1.6 - Jira issue types and hierarchy (see page 272)
 - 7.1.7 - Jira issues workflow (see page 279)
 - 7.1.8 - Personalized MS Outlook rule for Jira notifications (see page 287)
 - 7.1.9 - Creating custom HTML links to create Jira issues (see page 290)
 - 7.1.10 - Jira JQL hints, useful examples and managing filters (see page 293)
 - 7.1.11 - Jira common boards setup for cross-product collaboration (see page 306)
 - 7.1.12 - Jira Automation (see page 314)
 - 7.2 - Confluence (see page 318)
 - 7.2.1 - Confluence space structure (see page 319)
 - 7.2.2 - Documenting a Sprint Retrospective (see page 321)
 - 7.3 - Versioning and release notes (see page 323)
 - 7.3.1 - Documenting Release Notes (see page 332)
 - 7.4 - Dashboard metrics description (see page 339)
 - 7.5 - Logging time and transferring it to NetSuite ERP (see page 344)
 - 7.6 - User management (see page 355)

- [7.7 - Instruction to GDPR and threat assessment \(see page 364\)](#)
- [7.8 - Jira & Confluence permissions schema \(see page 373\)](#)
- [7.9 - ServiceNow integration \(see page 376\)
 - \[7.9.1 - Jira - ServiceNow integration board and issue flow \\(see page 377\\)\]\(#\)
 - \[7.9.2 - Jira - ServiceNow integration fields mapping \\(see page 380\\)\]\(#\)
 - \[7.9.3 - Jira - ServiceNow integration communication via comments \\(see page 384\\)\]\(#\)](#)
- [7.10 - \[FAQ\] Cloud to On-Prem migration \(see page 386\)](#)
- [7.11 - Jira Advanced Roadmaps \(see page 396\)](#)
- [7.12 - EG Security default schema for GDPR data access \(see page 407\)](#)
- [7.13 - AI Metrics for software development \(see page 410\)](#)
- [8 - Appendix \(see page 414\)
 - \[8.1 - Glossary \\(see page 414\\)\]\(#\)
 - \[8.2 - FAQ \\(see page 420\\)\]\(#\)
 - \[8.3 - Examples \\(see page 427\\)
 - \\[8.3.1 Retrospective \\\(see page 427\\\)\\]\\(#\\)
 - \\[8.3.2 Product and Sprint Goal \\\(see page 433\\\)\\]\\(#\\)\]\(#\)
 - \[8.4 - EG Agile Certification program \\(see page 437\\)\]\(#\)](#)
- [9 - Release notes \(see page 443\)
 - \[NGA\\(i\\) - version 3.0 \\(see page 444\\)\]\(#\)
 - \[NGA - version 1.1 \\(see page 445\\)\]\(#\)
 - \[NGA - version 1.2 \\(see page 446\\)\]\(#\)
 - \[NGA - version 1.3 \\(see page 447\\)\]\(#\)
 - \[NGA - version 1.3.1 \\(see page 448\\)\]\(#\)
 - \[NGA - version 1.4 \\(see page 449\\)\]\(#\)
 - \[NGA - version 1.5 \\(see page 450\\)\]\(#\)
 - \[NGA - version 1.6 \\(see page 451\\)\]\(#\)
 - \[NGA - version 1.7 \\(see page 452\\)\]\(#\)
 - \[NGA - version 1.8 \\(see page 452\\)\]\(#\)
 - \[NGA - version 1.9 \\(see page 453\\)\]\(#\)
 - \[NGA - version 1.10 \\(see page 454\\)\]\(#\)
 - \[NGA - version 2.0 \\(see page 455\\)\]\(#\)
 - \[NGA - version 2.1 \\(see page 456\\)\]\(#\)
 - \[NGA - version 2.2 \\(see page 457\\)\]\(#\)
 - \[NGA - version 2.3 \\(see page 458\\)\]\(#\)
 - \[NGA - version 2.4 \\(see page 459\\)\]\(#\)
 - \[NGA - version 2.4.1 \\(see page 460\\)\]\(#\)
 - \[NGA - version 2.5 \\(see page 461\\)\]\(#\)](#)

- NGA - version 2.6 (see page 462)
- NGA - version 2.7 (see page 463)
- NGA - version 2.8 (see page 464)
- NGA - version 2.9 (see page 465)
- NGA - version 2.10 (see page 466)
- NGA - version 2.11 (see page 467)
- NGA - version 3.1 (see page 468)
- NGA - version 3.2 (see page 469)

1 1 - Introduction

- 1.1 - EG Vision and goal (see page 26)
- 1.2 - Purpose of the playbook (see page 27)
- 1.3 - About the tooling (see page 28)
- 1.4 - About the processes (see page 29)

1.1 1.1 - EG Vision and goal

When we listen to our customers, we hear that in order to support them in being the best in their industry they need us to deliver at a faster pace. Delivery of features and functionalities that solve complex problems requires cross-functional collaboration with a high degree of transparency and knowledge sharing. Doing it efficiently and with high quality requires that we as a software company work in a harmonized way with the use of common processes supported by modern tools.

We also see that competitors are moving towards always updated software solutions where modern technologies help them continuously deliver functionalities, and the new generations of decision-makers expect modern and easily adaptable products that make their workforce efficient in their customer engagement.

To address these challenges we need to move closer in the EG family and build strong collaborative communities of practice within software development where our organization can easily be inspired and share the bright solutions that are created. We need to ensure that employees can share knowledge with colleagues across the company and we need to make it possible for employees to quickly become part of and contribute to another project or team in EG.

Vision for the Next Generation Agile program



The Next Generation Agile initiative, also called NGA, is driven by the vision to empower our organization to create and improve best-in-class software solutions and deliver at a faster pace to our customers.

This means that the CTO team will, in their work with the existing software teams and all new members of the EG family, strive to make them even stronger in their creation and delivery of valuable products to the market.

The dedicated Agile Coaches and DevOps Engineers will support and work with our organization to adopt a more common way of working based on well-proven agile frameworks and practices, which are supported by modern tools from planning and through the delivery pipeline. It will be as operational as possible, fuelled with reference points and best practices from both internal and external partners.

1.2 1.2 - Purpose of the playbook

The EG Agile Playbook, like the name suggests, illustrates *the rules of the game*. The playbook is intended to educate, inform and constitute a valid reference to how teams in EG should work with respect (but not limited to) Agile rules and principles in an everyday effort to deliver value. This playbook means to draw boundaries, describing mandatory, unified aspects of EG processes, as well as defining the room for individual interpretation and freestyle approach (see [2.3 - The common language \(see page 40\)](#)). Finally, this playbook is a mere attempt to define a common, consistent approach to software delivery within EG.

While the playbook consists of a sufficient amount of information for it to serve its purpose, it will remain a *living document*. As the EG organization learns to “play by the rules” of this document, it will continue to be verified and updated, taking into account observations and most valuable feedback. This is a commitment to empirical process control by **Inspection & Adaptation**. Like in all other areas, which this document addresses, it is crucial to follow the rule of *continuous improvement*; there is no exception in regard to the EG Agile Playbook which will grow to be more mature and robust with time.

1.2.1 Content of the playbook

The content of the EG Agile Playbook is very diverse and focused on providing the best possible way of exposing clear information and instructions concerning the particular area of the process. This is done by the means of:

- *guidelines* - to help understand the step-by-step execution of a part of a process
- *best practices* - to recommend or indicate the most proven approaches to a particular problem
- *references* - to support understanding of a particular concept by means of internal/external sources not directly included in the playbook
- *rules* - to explain the mandatory, common parts of the process for everyone
- *examples* - to provide additional materials meant to support execution of the playbook's rules & guidelines
- *values* - to educate on the meaning and benefits of the implemented principles & processes of the introduced commercial and custom frameworks, techniques, as well as methodologies

1.2.2 Target recipients of the playbook

The EG Agile Playbook is dedicated to everyone at EG, who is involved *in any way* in the production and delivery of value, frequently expressed by means of software and associated components. This target includes (but is not limited to) the following roles:

- software engineers
- infrastructure engineers

- architects
- product owners/product managers
- scrum masters/team leads
- line managers
- agile coaches
- business & UX analysts
- consultants
- product/business stakeholders

1.3 1.3 - About the tooling

As part of the transformation, a specific technological stack has been selected to support working in a new agile environment. These tools have been market-proven to be top-of-the-line enterprise technologies, supporting many leading global companies in day-to-day collaboration and agile software delivery.

In the table below, you can find a rough overview of the technological stack along with its intended use in EG.

Tool / Technology	Purpose & use in EG
 Jira Software	Workflow management & collaboration. Agile teams, as well as other stakeholders, should use Jira to communicate in reference to ongoing tasks and topics while providing an updated view on work progress. Jira is configured to comply with Agile principles, allow estimations and provide various means of collaboration across teams, including integration with CI/CD tools.
	Requirements management & collaboration. Confluence is closely integrated with Jira, thus allowing it to be extended with a requirements/documentation management platform to support the Product Backlog content within Jira. Scrum teams should use Confluence to view or edit (depending on role) the requirements documentation that is linked with a particular user story or epic, currently at work.

Tool / Technology	Purpose & use in EG
	Source code management. GIT delivers fast, distributed version control. GitHub is the first block in Continuous Integration flow in EG. It's tightly integrated with Jira: allows creating new branches straight from the Jira issue and gives detailed source code-related overview within the issue in Jira (branches, pull-requests, merge statuses, and others).
 Microsoft Teams	Notifications & team communication. This is the official channel for automated notifications from the CI/CD pipeline to Agile teams, as well as an open, collaborative platform for Agile teams to interact on a daily basis. The tool facilitates communication, especially for distributed teams across EG locations.
	Enterprise social network platform. Yammer continues to be the EG-wide platform for inner company communication and socializing. It allows company users to broadcast valuable information, achievements, or queries on a common forum.

1.4 1.4 - About the processes

In addition to standardized tooling across EG, the transformation implies the introduction of new, unified processes to enable the delivery of valuable software in accordance with Agile principles and DevOps best practices.

The processes being referred, are based on a number of frameworks and methods, with several modifications and custom-fitted approaches included to provide the best fit to EG company organizational structure and at the same time enforce self-organization at a team level. The frameworks mentioned, which build the foundation of the EG Agile product delivery process portfolio, include the Scrum framework (based on the Scrum Guide - Scrum.org¹), the KanBan method (based on the KanBan University - kanban.university), Small Scale Scrum (based on AgileAlliance Experience Report by Leigh Griffin & Agnieszka Gancarczyk) and a number of custom, specialized solutions to accommodate consultancy or high-granularity specific agile teams in EG.

The sole proposed solutions, as well as their combinations will be used to provide the best fit for the BU during the onboarding. Having a palette of possibilities in regards to the processes, allows the NGA agile coaches to adjust the target solution as close to the BU needs as possible, resulting in the most optimal process implementation.

¹ <http://Scrum.org>

1.4.1 Scrum



Scrum is a lightweight framework intended for agile delivery of complex products. Scrum is particularly proven extremely efficient (when correctly implemented) for delivering value through software-oriented deliverables.

Scrum will serve as a baseline for the majority of the software delivery processes at EG, beginning from requirements engineering to production deployment.

Benefits and implications of working with Scrum

- complete transparency → faster identification of impediments, bottlenecks, and blockers
- an iterative approach → focusing on delivering value as soon as possible, inspecting and adapting
- close collaboration → hear and be heard to provide valuable feedback throughout the delivery process
- team ownership → the Scrum team is responsible for the delivery of a product, with support from the organization
- mutual respect → is the foundation, on which high performing and happy teams are built

1.4.2 KanBan



The KanBan method is a set of observed principles and practices which have proven in successful KanBan implementations all over the world. KanBan will be the framework of choice when considering a greater focus on throughput of the flow in optimizing streamlined, continuous software delivery or service/maintenance work.

The communication scheme in KanBan fosters building a network of ceremonies that reflect all the needs on all levels of the organization. The scheme should follow the principles of inspection and adaptation all along the way, as in case of all agile delivery approaches - the cadence should be continuously reviewed and optimized to deliver the most effective process.

Benefits and implications of working with KanBan

- flow visualization → making the process transparent to identify its clogs and bottleneck
- focus on completion of work → raising productivity by limiting work in progress
- improve on past learnings → using past data to alter and optimize the flow of work

- boost team collaboration → fast feedback loops and experimental evolution of the process
- transparent policies → align, agree and follow clear rules for processing of backlog items

1.4.3 Small Scale Scrum

Small Scale Scrum

Small Scale Scrum is a lightweight interpretation of the Scrum framework for agile delivery small products (product increments) in a small team environment. This approach proves to be very valuable in situations where the project and/or team have limitations not allowing full utilization of the Scrum framework.

EG Scrum is the leading candidate for the implementation of Small Scale Agile for many reasons including its popularity, developers' preference, high success rates for Scrum adoption and project deliveries, and strong principles and values including focus, courage, openness, commitment, and respect.

Small Scale Scrum can be best described as “a people-first framework defined by and for small teams (a maximum of three people) and supporting planning, developing, and delivering production-quality software solutions.” The proposed framework centers around the concept of team members occupying multiple roles on any project. Small Scale Scrum is valuable due to its strong support for the small, distributed teams found in organizations all over the world.

Small teams need new ways to meet customers' continuously growing expectations for rapid delivery and high quality, and Small Scale Scrum's guidelines and principles help address this challenge.

Small Scale Scrum is based on the Scrum framework. The primary difference is the timing and flexibility of events. You can read about more details in the next chapters dedicated to the specific events.

Benefits and implications of working with Small Scale Scrum

- value-based communication → to accommodate for a limited time and resources all communication should bring value
- flexible roles and events → scrum roles can be rotated/shared while events can be selective to reduce overhead
- quick iterations → focus on more frequent validation of the work to improve quality and inspect the PoC
- customer growth focus → ensuring the customer grows together with the solution to provide the best possible outcome

1.4.4 NGA Custom approaches



NGA introduces processes for establishing a dedicated approach to **consultancy teams**, who focus around billable hours as well as teams with high product granularity. The practices and techniques in scope, aim to trim and mold the default EG Scrum framework allowing specific contexted teams to make the most out of agile product delivery.

Consultancy teams will benefit from an individual alignment during an evaluation process, to determine the best way of working within the teams.

High-product-granularity teams will benefit from common techniques, such as pair-programming, altered backlog refinement sessions to allow more focus on specialization areas rather than involvement of the entire team.

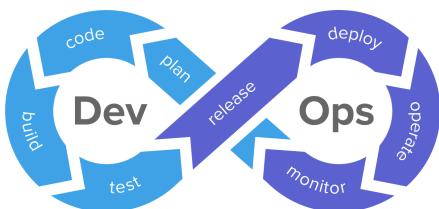
High-product-granularity teams are characterized with a dispersed focus either on multiple areas with high specialization differentiation within the team, or with working on multiple independent (smaller) products at the same time. This setup requires the Scrum events or KanBan events to be altered in a way which will drive the team into a process of reducing the bus factor on the team, while at the same time supporting their existing product delivery.

In the chapters dedicated to the specific events, you will read about introducing techniques, which are dedicated to teams where their personal competencies need to be acknowledged but also strive to become a well-functioning unit rather than a group of individuals.

Benefits and implications of working with NGA Custom approaches

- custom-fit approach → best solution available for the specific team, to raise process utilization and satisfaction in the teams
- eliminate waste → maximize productivity while accomodating the core of the agile frameworks
- agile mindset → focus on an agile product delivery process that re-inforces the agile mindset and builds on top of it

1.4.5 DevOps



Besides Agile transformation, EG will shift more into DevOps way of working. DevOps is a set of practices that combines software development (Dev) and information-technology operations (Ops) which aims to shorten the systems development life cycle and provide continuous delivery with high software quality. It's the practice of operations and development engineers participating

together in the entire service lifecycle, from design through the development process to production support.

The DevOps toolchain includes:

- Coding – code development and review, source code management tools, code merging
- Building – continuous integration tools, build status
- Testing – continuous testing tools that provide quick and timely feedback on business risks
- Packaging – artifact repository, application pre-deployment staging
- Releasing – change management, release approvals, release automation
- Configuring – infrastructure configuration and management, infrastructure as code tools
- Monitoring – applications performance monitoring, end-user experience

The first step in EG's DevOps journey will be the Continuous Integration - the practice of merging all developers' working copies to a shared mainline as often as possible and trigger the automated build.

2 2 - Overview

- 2.1 - The BIG picture (see page 34)
- 2.2 - The Agile Manifesto and change of mindset (see page 39)
- 2.3 - The common language (see page 40)
- 2.4 - Company & Scrum values (see page 42)

2.1 2.1 - The BIG picture

Having the understanding of the goal of this playbook, as well as an introduction to the tooling and processes in scope, it's worth taking a look at the big picture - how it all comes together. The core of Next Generation Agile is founded on top of the EG Scrum framework based on the Scrum.org Scrum Guide. However, as the organization grows, the needs for a broader portfolio of possibilities dedicated to agile product delivery increase. In order to accomodate those parts of the organization, which could not benefit as much from EG Scrum as the majority, there are a few other approaches available in EG to match the style of the team. These approaches aim to address the following characteristics:

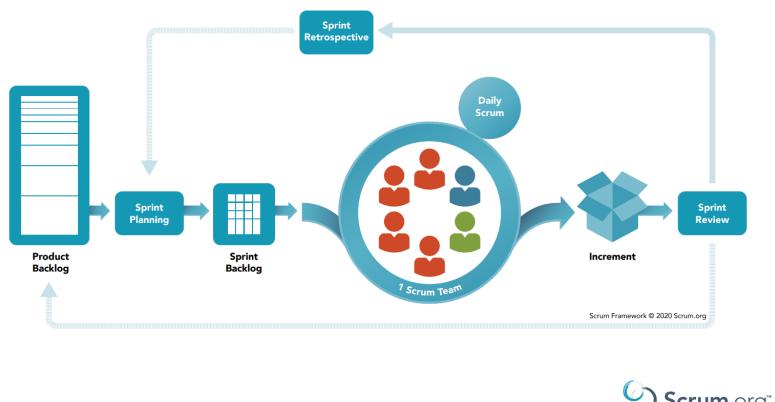


- **cross-team & cross-product collaboration**

- **maintenance focus**
- **high product granularity & high specialization**
- **consultancy & billable hours**
- **small sized teams**

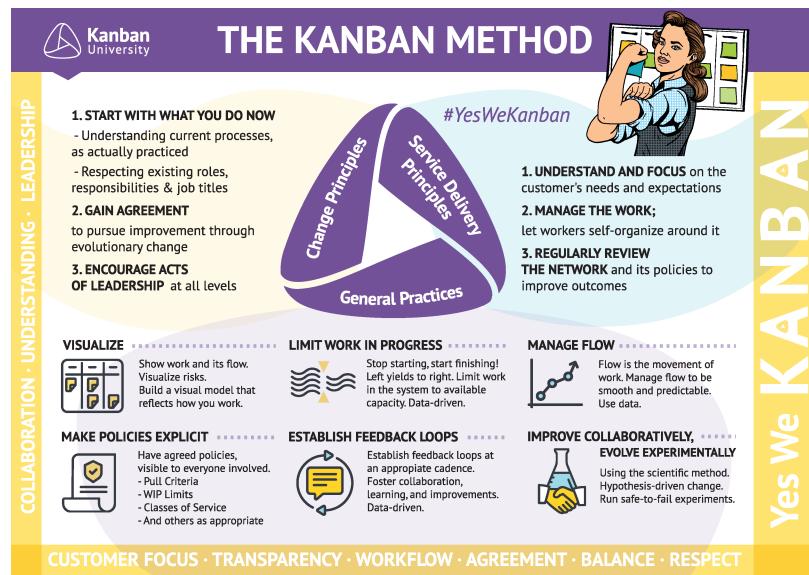
In order to support the diverse environment of product delivery in EG, KanBan, Small Scale Scrum and a set of available & custom best practices and techniques are incorporated into the NGA to provide the most optimal setup for the product teams in EG. Along with the dedicated Agile Coach, the best approach and composition of frameworks and methods will be established based on the initial evaluation of the team/BU characteristic.

SCRUM FRAMEWORK



(The Scrum sprint cadence infographic - <https://www.scrum.org/>²)

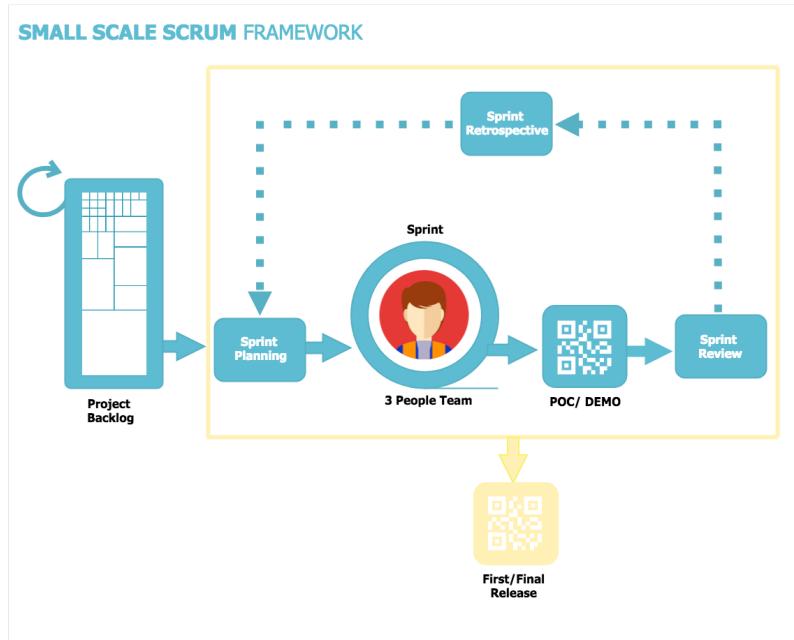
² <https://www.scrum.org/>



(The KanBan method infographic - <https://kanban.university/>)



(Next Generation Agile - The EG Agile Playbook (see page 22))



(Small Scale Scrum infographic - [Small Scale Scrum](#)³)

Product focus

The overall delivery focus is shifting to be more product-centric and product-oriented. In the end, it is important that EG customers are satisfied with the products that are being delivered to them. In order to understand and add value to the product, it is defined by a single Jira site, a single Confluence site and of course, a single Product Backlog ([more in 5.3 - The Product Backlog \(see page 160\)](#)) across the entire EG organization. This is both the starting and finishing point of a journey to build and enhance EG products.

The Product Backlog, defined in Jira is supported by the content in Confluence, including product requirements and documentation. As a starting point, the teams focus on what is to be delivered, when and with what value to the product. The finish summarizes if that goal was accomplished and the expected value delivered, which in turn leads to a decision whether the increment should be launched to production. All this is supported by appropriate artifacts and metrics ([as described in 5 - Managing and using artifacts \(see page 150\)](#)) to enforce Transparency (one of 3 Scrum pillars) throughout the processes.

2.1.1 Process phases

A consistent end-to-end agile software delivery process enables EG to:

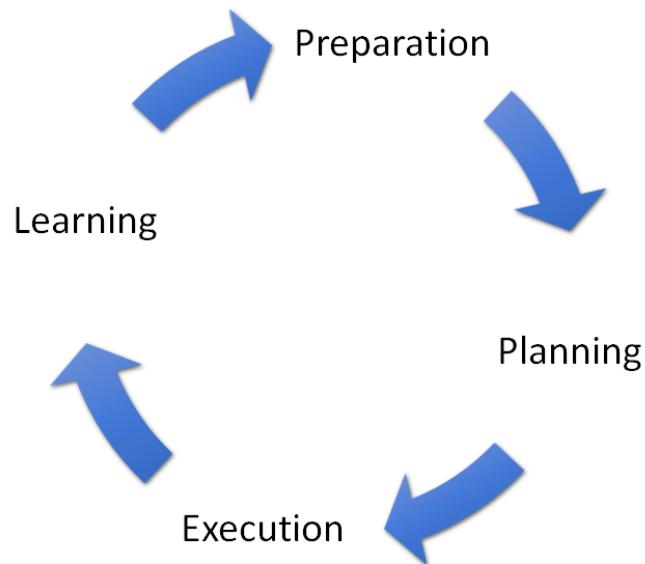
- remain capable of delivering world-class software at a fast pace
- empower and be empowered while responding to change
- utilize its full potential and focus on delivering value

The EG processes are built around a portfolio of agile frameworks and methods with Scrum as the default recommendation. NGA aims to enable complete collaboration of agile teams throughout these processes while maintaining compliance with EG governance. Particular roles ([as described in 3 - The team, roles and responsibilities \(see page 47\)](#)) will be responsible for fulfilling their duties in scope of all events, whether we talk about Scrum sprint cycle, KanBan flow or a dedicated combination of techniques ([as described in 4 - Events and working together in sprints \(see page 107\)](#)). Despite the selected approach, the process for product

³ <https://www.agilealliance.org/resources/experience-reports/small-scale-scrum/>

delivery in EG should aim to be incremental, based on empiricism and should be transparent. All those characteristics fit into the agile mindset, whether it's Scrum's pillars or KanBan's practices to improve and evolve.

The following phases in the process can be distinguished:



1. **Preparation** - all activities associated with building and managing the backlog and high-level product roadmap (i.e. Product Backlog refinement)
2. **Planning** - deciding what and when should be delivered, defining sprint and release scope (i.e. Sprint Planning in EG Scrum)
3. **Execution** - elaborating on how and doing the work, followed by a decision to accept the work or not (i.e. task breakdown, the Sprint in EG Scrum)
4. **Learning** - (probably the most important phase when thinking about continuous improvement) verification of accomplished goals, looking at what was done well and what could be improved (i.e. Sprint Retrospective in EG Scrum)

Naturally, to bring all of this to life, it is crucial to remain open to change and new learnings while strengthening the collaboration within and across the teams. Communication is a vital element of the puzzle and it is one of the main expectations towards you all when thinking about EG's BIG Picture.

2.2 2.2 - The Agile Manifesto and change of mindset

A very important part in shaping and cultivating our organizational culture plays the Agile Manifesto, which being the foundation of Agile Software Development, represents the attitude and mindset that should accompany everyone on a daily basis.

The Agile Manifesto was the result of a gathering of representatives of various alternative (modern) and lightweight development processes, including Scrum's Ken Schwaber and Jeff Sutherland (Scrum.org⁴ founders). The Agile Manifest represents values that very well complement all of the frameworks, methodologies, and other agile techniques in the scope of this playbook, introducing an innovative way of thinking about product development.

2.2.1 **We are uncovering better ways of developing software by doing it and helping others do it.**

Through this work we have come to value:

2.2.2

**Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan**

2.2.2.1 *That is, while there is value in the item on the right, we value the items on the left more.*

Agile does not only represent a group of approaches and principles. Agile is a philosophy and it can be a way of life. Agile aims to foster a particular mindset that is so crucial to the successful outcome of any undertaking. The best way to explain that is to elaborate on the 4 fundamental statements:

2.2.2.2 Individuals and interactions...

Focus on people and the interactions between them first. While tools and processes are useful, very often making our everyday work significantly simpler, they can be faulty and misleading in complex projects. What is key, is that tools and processes should result from the decisions of a group of people, rather than the other

⁴ <http://Scrum.org>

way around: having tools and processes impact the way that a group makes decisions. People always come first.

2.2.2.3 Working software...

While the need for good documentation should not be underestimated, the measure of success is always the resulting software with its quality and value to the customer first. Creating any kind of documentation should support the value of a product that is being built, but its shape nor content can never endanger the delivery or resulting value of that product. With documentation as input, it always needs to be verified that a document is coherent with the product vision and provides added value.

2.2.2.4 Customer collaboration...

Contract negotiations can often lead to situations where the very object around which they are built (the ordered Product) is negatively affected through contractual decisions. It is understandable that contracts are needed to formalize agreements, especially between unfamiliar entities; however, rather putting the focus on building a good relationship with the counterpart and working in an atmosphere full of trust, will impact the negotiation to be less relevant. This in the end will lead to a more productive collaboration to enrich the outcome value of the product, leaving the contract behind as a simple necessity with basic facts. It's impossible to predict everything that may happen within a contract, hence it's good to remain open to change instead of spending time on managing such cases.

2.2.2.5 Responding to change...

The previous statement was summarized with a mention of openness to change, while this one is about responding to it. It's a great habit to plan ahead and most of the products will face the need of having a roadmap or preparing a yearly/quarterly plan. However, in order to comprehend the meaning of this statement, it's mandatory to acknowledge the fact that the environment around us changes quickly and frequently. Thus it is crucial to stay open to new learnings, even if that means changing a part of a feature that has become obsolete. Much more value will come out of it if the feature is redefined than if it's delivered according to the originally intended plan.

2.3 2.3 - The common language

One of the main goals of the ongoing transformation is to establish a *Common Language*, which will define how we work and deliver software products throughout the entire EG company. A unified, common language will help to build a coherent understanding and working culture across all Business Units, which will lead to a closer integration within EG; this will allow delivering better products faster, giving the EG advantage in the contemporary, dynamically changing market.

The common language is present in the entire EG Agile Playbook, defining the rules, guidelines, and values that can be distinguished into the mandatory and the freestyle.

2.3.1 The Mandatory

The mandatory parts of the common language visible across the playbook specify the sets of tools, processes, rules, and ways of working which define the EG way of getting things done. This constitutes the sole part EG's identity which must be common for everyone and is defined globally.

As an example, the new Jira issue workflow, as well as Jira Software itself, could be considered as the mandatory part of building EG's common language. Jira is EG's official issue management system for agile software development and no other tool should substitute its purpose. Along with Jira, the defined common Jira workflow (*as defined in Jira - issues workflow (see page 279)*) is also a cross-EG official flow for issue management from an idea to a releasable product increment.

2.3.2 The Freestyle

The freestyle part of the common language, which also will run its way throughout the playbook, specifies the areas which are more unrestricted and less specific, allowing a more freestyle interpretation by particular teams or units. It will provide guidelines, ideas, suggestions or a spectrum of options, but will not illustrate an official, single way of working in that particular area or topic.

As an example, the Confluence page documents structure is quite flexible to some extent, allowing teams to customize the content setup according to their product-based needs. An additional example is selecting the scenarios for running a Sprint Retrospective (*see 4.6 - The Sprint Retrospective (see page 126) & 6.3 - Examples (see page 427)*). The fact of a necessity to run a retrospective would fall under the category of the mandatory part of the common language in case of teams working with the pure EG Scrum approach, however, choosing a scenario template or working out a custom one is freedom given by the common language's freestyle.

2.3.3 CTO Common Language definition

While attempting to define a more tangible representation of the EG Common Language, the following listing was built, to help visualize its most relevant elements:

2.3.3.1 Mandatory elements of the EG Common Language

- backlog management tool: Jira with default EG configuration
- content mgmt./release notes tool: Confluence with default EG configuration
- source code management tool: GitHub
- build automation tools: GitHub Actions (preferred. Option to build on the Cloud, on-premise dedicated agent or Kubernets runners farm), Jenkins or Azure DevOps Pipelines (based on existing tooling/CTO recommendation)
- test management tool: Zephyr Scale
- designated Product Owner & Product Manager role for each product (1 person can share both - most commonly for small products)
- designated facilitator (Scrum Master, Flow Master, etc.)
- 1 EG main product per Jira project / Confluence space
- an iterative approach to product development with a CTO approved form of planning, demo, evaluation (excluding strictly support/maintenance teams)

- EG compliant definition of Ready & Done
- use of written english in daily written communication and documentation within the abovementioned tools

2.3.3.2 Recommended elements of the EG Common Language

- consistent GIT branching model approved by CTO
- quality gates: SonarQube, OWASP Dependency Track, Trufflehog Secrets Scanning
- pull-requests with at least one approver
- continuous integration
- EG Scrum framework (especially for less mature teams)
- stable agile teams without shared members
- no cross-product dependencies

2.4 2.4 - Company & Scrum values

In EG we look on our work through the lens of three values: Customer Focus, Deliver what we promise and Respect for each other. You will find them described below in details with examples of how those value are represented in day-to-day work. EG Scrum is the default approach to agile software development among the remaining available agile frameworks and techniques in EG. Therefore, it is worth taking a look at the Scrum framework, which is founded on five values that all-together support the company values. In the second section you will find a brief description of the Scrum Values and how they mach to overall picture.

2.4.1 Company values

Customer Focus We have deep knowledge of our customers business and industries 	Deliver what we promise We take responsibility and deliver mission critical solutions 	Respect for each other We honor diverse ways of thinking 
---	--	---

2.4.1.1 Customer Focus

We have deep knowledge of our customers business and industries

We enable our customers to become industry leaders by providing best-in-class vertical software and support. Our software is infused with the highly specialized knowledge of all of our employees. We always strive to get better: We are a team of innovators, problem solvers, and doers. Without our customers and bright employees, no EG.

Examples of how this value is shown in EG

- We go the extra mile to support our customers and keep their business running.
- We always prioritize customer inquiries over internal matters.
- We quickly resolve issues that customers report.
- We listen to customer's wishes through user groups, communities, and 1-1 contact and consider these in our product development.
- We keep our self updated about the latest trends and demands in the industries and use this knowledge to improve our solutions.
- We are ambitious on behalf of our customers – we challenge them, each other, and our self.
- We dare to ask questions and be curious to gain greater knowledge.

2.4.1.2 Deliver what we promise

We take responsibility and deliver mission critical solutions

It is an integral part of our culture that we take responsibility for our contribution to customers, to each other and to the society we live in. We deliver quality and reliability and we aim to always be a trusted partner. We know that the solutions we deliver are mission critical for our customers and that we take very seriously.

Examples of how this value is shown in EG

- We are reliable, our customers trust us and are happy to stay with us.
- We trust our colleagues and count on each other to deliver value.
- We have integrity and deliver what we promise and if not, explain why and when we will deliver.
- We secure alignment with customers and colleagues so we all know what is expected and agreed.
- We care deeply about quality and understand the customers need for quality.
- We are pragmatic and agile in our way of working.

2.4.1.3 Respect for each other

We honor diverse ways of thinking

In EG we embrace workforce diversity and value a diversity of perspectives – leveraging the diverse personalities, thinking, skills, experience and working styles of our employees, customers and other stakeholders. Scandinavian leadership culture in everything we do: No politics. Only real responsibility and collaboration.

Examples of how this value is shown in EG

- We have many different types of customers. Respecting diversity and different viewpoints often gives the best solution.
- We respect and accept diversity in our teams and see this as an advantage.
- “We-first” instead of “me-first” is an integral part of our culture and way of collaborating.
- We see each other as equal team members, regardless of position or other factors and everyone is entitled and expected to have an opinion.
- We care for each other to ensure that we all are able to perform at our best.

We have fun at work and believe that respect and informal communication can be balanced.

2.4.2 Scrum Values and their relation to company values

Below you will find five Scrum Values which are the foundation of the framework together with a brief description of what they mean. If you want to dive deep into the topic, please refer to the link in the source.



| Source: <https://www.scrum.org/resources/blog/5-scrum-values-take-center-stage>



The scrum team **focuses** on the Sprint and Sprint goal delivery which are closely related to the most relevant and urgent client demands. Furthermore, it's even more beneficial to the Customer when the Scrum Team is **open** to the different points of view and approaches them with curiosity. When the team has the **courage** to challenge some assumptions and propose some of their own that they believe can bring even more value, the Customer benefits with an even greater product.



The Scrum Team is **committed** to achieving Sprint goals that are directly co-related to the Customer's needs. They **openly** and transparently share their progress of the work and if needed they align to the current needs.



Scrum has the same value **Respect** which is understood in a similar way and supporting the team working and team spirit in the journey of software delivery.

As you can see the above examples are closely coupled with our Company values. They are not fulfilling the portfolio of possibilities and situations in which multiple Scrum values are supporting our EG values. They go hand-in-hand and don't exclude one another.

3 3 - The team, roles and responsibilities

- 3.1 - What an EG scrum team looks like and how it operates (see page 47)
- 3.2 - The role of Developers (see page 51)
 - 3.4.1 - The role of a Business Analyst / UX Designer (part of Developers) (see page 58)
 - 3.4.2 - The role of SW Architect (see page 61)
 - 3.4.3 - The role of QA Tester / Engineer (see page 69)
- 3.3 - The role of a Product Owner and Product Manager (see page 77)
- 3.4 - The role of a Scrum Master (see page 83)
- 3.5 - The role of an Agile Coach (see page 88)
- 3.6 - The role of a DevOps Engineer (see page 90)
- 3.7 - The role of a Manager (see page 93)
- 3.8 - The role of a Consultant (see page 97)
- 3.9 - EG Scrum roles in various contexts (see page 99)

3.1 3.1 - What an EG scrum team looks like and how it operates

Everything starts with building the right team for the job. Having a proper team setup is a big step towards building a successful product. Things that should be considered when constructing a team, are:

- **Does the team have all the necessary competencies, tools, and skills to do the work and deliver?**
In other words, is it cross-functional? Full-stack is nice to have but not necessarily required. However, the team should be composed of members who have all the means needed to contribute to building the product.
- **Is the team not too big nor too small?**
An ideal team should be between 5 and 10 team members, including a 1 Scrum Master, 1 Product Owner, and 3 - 8 Developers (software engineers, tester, consultants, analysts, designers).
- **Is the team independent of external influences?**
The team should be able to perform their work with as little external dependencies as possible; all outer dependencies should be identified, maintained and managed by the team.
- **Do the team members get along?**
Working together in a team is best when both the workplace and personal relationships are on a high, positive level. It's not always possible to expect such a scenario, but being together in a team requires team members to be respectful and open to each other.
- **Are the team members fully co-located or fully distributed?**
Teams work best when sitting next to each other - that is the most efficient way to communicate. While that is not always possible in a contemporary enterprise model, one should seek to construct a fully distributed team setup. It's important to avoid mixed setups as they are most impeding to the team's communication. Building fully distributed teams can have its benefits in the availability of people with relevant skills to be part of an agile team worldwide.
- **What are the interferences with the external roles (i.e. Consultants)?**

The relationship and alignment of the collaboration with the roles that are not in Scrum, but are interfering with the Scrum Team are also important. This may vary in the specific Business Unit setup, but finding the best possible way to work together is essential for the flow of the work, delivery rate as well as customer care.

NGA 2.0

In NGA 2.0 we can select the best option for the team:

- EG Scrum
- Kanban
- Small Scale Scrum

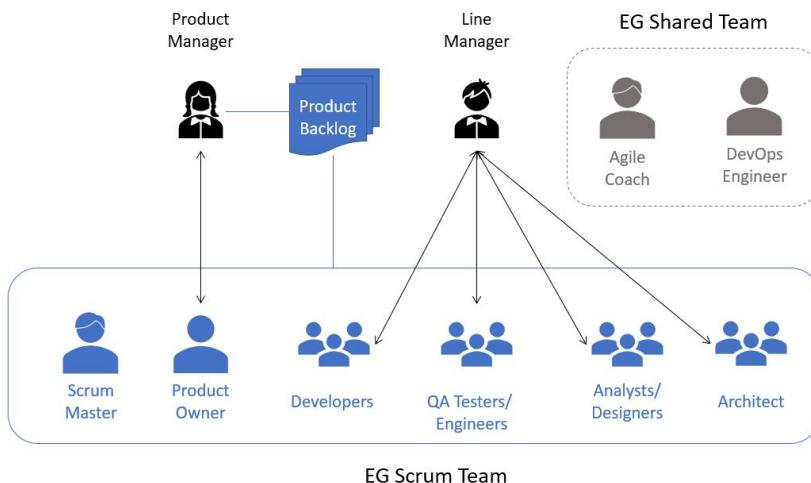
And also we have gathered here best practices for teams who are struggling with:

- Consultancy Teams
- High Granularity Teams

The above approaches will help Agile Coaches and Business Units to fill out their needs in the Agile way of working.

More about building a team and about team-building exercises can be found: [4.8 - Team building \(see page 136\)](#)

3.1.1 The EG Scrum Team environment



An EG Scrum Team consists of:

- **Developers** - more in [3.2 - The role of a Developer \(member\) \(see page 51\)](#)
- **a Product Owner** - more in [3.3 - The role of a Product Owner and Product Manager \(see page 77\)](#)
- **a Scrum Master** - more in [3.4 - The role of a Scrum Master \(see page 83\)](#)

The responsibility for delivering a product (potentially releasable increment - more in [5 - Managing and using artifacts \(see page 150\)](#)) lies with Developers. Nevertheless, they are supported by both the Scrum Master and Product Owner from their Scrum Team, as the members of the EG Shared Team:

- **Agile Coach** - more in [3.5 - The role of an Agile Coach \(see page 88\)](#)

- **DevOps Engineer** - more in [3.6 - The role of a DevOps Engineer \(see page 90\)](#)

In addition to operational support provided by the EG Shared Team, some of the EG Scrum Team members also benefit from support on another level from their **Line Managers** (more in [3.7 - The role of a Manager \(see page 93\)](#)). Their main focus is to help resolve impediments escalated by the Scrum Masters and mentor Developers on self-management and continuous improvement.

The **Product Manager** (more in [3.3 - The role of a Product Owner and Product Manager \(see page 77\)](#)) will collaborate closely with the Product Owner on building a proper Product Backlog (more in [5.3 - The Product Backlog \(see page 160\)](#)) which best represents the product vision and expectations of the stakeholders (customers).

The **Consultant** (more in [3.8 - The role of a Consultant \(see page 97\)](#)) - is collaborating with the Scrum Team and especially the Product Owner with the requirements gathering and clarification, refinement process, customer relationships, engaging the customers in requirements clarification, reporting the bugs.

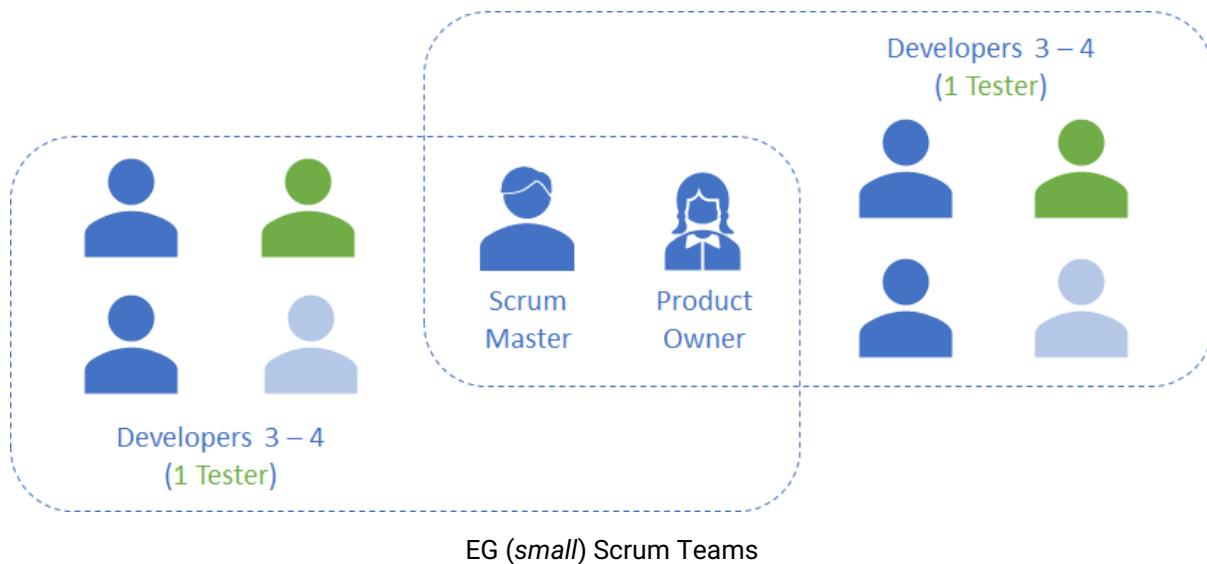
Consultants are sometimes aligning with the Product Owner in the Product Backlog prioritization process (keeping in mind that the Product Owner is accountable and decisive here). In some setups, the Consultants are having the Developer shared role in the situations, where they are participating in the development process (i.e. testing) or Product increment creation (i.e. release and configuration process).

3.1.2 EG Scrum Team types and detailed role distribution



EG (*full*) Scrum Team

Developers in a **full** EG Scrum Team are 5-8 team members representing roles necessary to deliver a product increment at the end of a Sprint. These are usually roles of **Software Engineers, System Engineers, Automation and Testers**. Less frequently and in a much lesser capacity, these roles also include **business analysts** or **UX designers** (more in [3.4.1 - The role of a Business Analyst / UX Designer \(see page 58\)](#)). The full EG Scrum Team is completed with a dedicated Scrum Master and Product Owner.



Developers in a **small** EG Scrum Team consists of 3-4 team members which are usually only **Software Engineers** and **Testers**. While it is not impossible, it is uncommon that such small teams will dedicate a separate role for a business analyst or a UX designer. Hence, the teams are often purely technical and dedicated to a specific product with narrow specialization.

The ratio of manual testers in an EG Scrum Team depends on the number of **Software Engineers** and the level of test automation within the product.

- smaller teams will most likely require 1 manual tester, while larger ones will require 2 per team
- larger teams with a good test automation approach will most likely require 1 manual tester, assuming that there exist test automation skills within the team

3.1.3 The Kanban team

Kanban is and remains the “start with what you do now” method, where initially no one receives new roles, responsibilities, or job titles. So there are no required roles in Kanban and the method does not create new positions in the organization. However, those roles have emerged from common practice in the field and are now defined in the method itself. It is the purpose of the roles that is important, rather than assigning someone a job title, so it may be helpful to think of the roles as “hats” people wear in carrying out these functions:

Product Owner - in Kanban team is responsible for understanding the needs and expectations of customers, and for facilitating selecting and ordering work items at the Planning Meeting. See more in [3.3 - The role of a Product Owner and Product Manager \(see page 77\)](#)

Scrum Master (or alternatively Flow Master) - in Kanban team is responsible for the flow of work in delivering selected items to customers and for facilitating the (daily) Kanban Meeting and Delivery Planning. See more in [3.4 - The role of a Scrum Master \(see page 83\)](#)

Developers -in Kanban team are responsible for creating a predictable flow of value. See more in [3.2 - The role of a Developer \(member\) \(see page 51\)](#)

3.1.4 The Small Scale Scrum

Small Scale Scrum can be best described as “a people-first framework defined by and for small teams (a maximum of three people) and supporting planning, developing, and delivering production-quality software solutions.”

The Three-People Team is comprised of the development team. The development team is expected to be involved in gathering and clarifying business requirements, doing development work, performing quality testing, and releasing software to the customer.

A Small Scale Scrum consists of:

- **Developers** - more in [3.2 - The role of a Developer \(member\) \(see page 51\)](#)
- **Product Owner and Scrum Master** - we do not distinguish that role in this approach. Small Scale Scrum is based on the proper communication and cooperation between small team members.

3.2 3.2 - The role of Developers

NGA 2.0

In NGA 2.0 can be selected the best option for the team:

- EG Scrum
- Kanban
- Small Scale Scrum

And also we have gathered here best practices for teams who are struggling with:

- Consultancy Teams
- High Granularity Teams

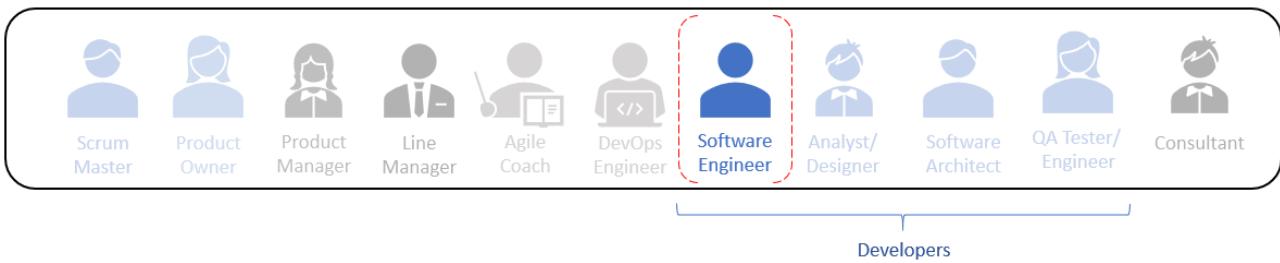
The above approaches will help Agile Coaches and Business Units to fill out their needs in the Agile way of working.

Developers are the people in the Scrum Team that are committed to creating any aspect of a usable Increment each Sprint.

The specific skills needed by the Developers are often broad and will vary with the domain of work. However, the Developers are always accountable for:

- Creating a plan for the Sprint, the Sprint Backlog;
- Instilling quality by adhering to a Definition of Done;
- Adapting their plan each day toward the Sprint Goal; and,
- Holding each other accountable as professionals.

The Scrum Guide, SCRUM.org



3.2.1 Developers should strive for being:

- Self-managing - the team needs to evaluate and decide on their own how to work together on delivering the potentially releasable product increment every sprint. The team may seek council outside of the team if necessary, to obtain specific input required to progress towards the sprint goal (such as Architect, DevOps Engineer, etc.), however, such external dependencies should always be minimized. No one should instruct the team on how to execute their tasks otherwise.
- Cross-functional - the team needs to possess all known skills, competences and tools with respect to the product backlog which is known to be processed at a given moment. The team may be composed of specialists in particular domains, who as a group contribute to building the product cross-functionally, however, the team should aim to distribute such skills and knowledge throughout the team to avoid bottlenecks. Should a need for new competencies arise, the team should plan to gain the appropriate tools and skills in the scope of their sprint, in order to complete the sprint backlog items.
- Flat-structured - the team does not recognize any titles nor statuses within it. All members of the team are equal regardless of the executed tasks and all members share equal accountability for the end result. The team's size should fit between 3 and 8 members including all roles necessary for a cross-functional delivery of the product increment.

3.2.2 Responsibilities and duties of a Developer (team member)

1. Refining, breaking down and estimating Product Backlog items
 - a. Learning the goal and value of Product Backlog items
 - b. Providing feedback and proposing changes to the Product Backlog items and the Product Backlog
 - c. Decomposing epics into smaller epics or user stories (more in [5.3 - The Product Backlog](#) (see page 160) and [4.2 - The Backlog Refinement](#) (see page 113)) and user stories into sub-tasks (more in [4.3 - The Sprint Planning](#) (see page 117))
 - d. Estimating epics high-level and user stories, using Story Points or other agreed methods (more in [5.2 - Refining and estimating an initial Product Backlog](#) (see page 152))
 - e. Handling incoming issues from ServiceNow in terms of initial evaluation to support the Product Owner with input to the decision regarding placement of the issue in the Product Backlog
 - f. Communicating with ServiceNow support to acquire any necessary data, that is not in scope of the teams' competences, for handling an incoming ServiceNow support issue and further assessment

- g. Identifying components/services that the team could reuse or that could be reused by other teams (more in [Shared Components⁵](#))
2. Working on delivery of a potentially releasable product increment
- a. Working on sprint backlog items with respect to their complexity and priority
 - b. Verifying the validity of the sprint goal on a daily basis
 - c. Striving for delivery of a releasable product increment at the end of the sprint
 - d. Designing, implementing and testing solutions as part of the product
 - e. Release Notes is document containing information of new features, changes in the products or product updates. During release of usable product or part of it, developers have to prepare release notes. Full instruction how to do them is in the chapter [6.3⁶](#) and [6.3.1 \(see page 332\)](#)
3. Enforcing transparency through the use of EG agile software development tooling stack
- a. Using Jira for development workflow management and specifically scrum boards, as well as dashboards for transparent work progress visualization
 - b. Using Confluence for documentation
 - c. Using GitHub as the code repository
 - d. Keeping the state of work progress and documentation up to date in the mentioned tools
 - e. Use and contribution to Backstage for reuse of components/services ([Shared Components⁷](#))
4. Participating in all Scrum Events
5. Enforcing inspection and adaptation, as part of continuous improvement
- a. Seeking solutions to malfunctioning processes within the team and implementing them
 - b. Checking own work for quality, consistency and coherence with the sprint/release goal
 - c. Looking for improvements in tooling and practices to improve the work of the team and implementing them
 - d. Providing feedback to the Scrum Master and suggesting improvements in any other area of the team
6. Striving for increased self-management and autonomy
- a. Learning from Agile Coaches, Scrum Master and this playbook about proper implementation of Scrum in EG and following the guidelines
 - b. Aiming for achieving an autonomous team setup with fully dedicated team members
 - c. Taking ownership of Scrum Events, sessions, metrics and artefacts when external support is unnecessary
7. Using metrics and artefacts as intended
- a. Owning the sprint backlog and working on the product backlog
 - b. Checking and confirming the Definition of Ready (more in [5.5 - The Definition of Ready \(see page 176\)](#)) as well as the Definition of Done (more in [5.6 - The Definition of Done \(see page 180\)](#)) for Backlog items
 - c. Keeping team information, metrics, rules and agreements up to date (more in [5.8 - Team page \(see page 211\)](#))
 - d. Using velocity and capacity (more in [5.7 - Using team metrics \(see page 184\)](#)) as input in evaluating the sprint and release plans

⁵ <https://confluence.eg.dk/display/NG/Shared+Components>

⁶ <https://confluence.eg.dk/display/NG/6.3+-+Versioning+and+release+notes>

⁷ <https://confluence.eg.dk/display/NG/Shared+Components>

3.2.3 RACI matrix for EG Scrum events and sessions

Backlog refinement	R	Developers are responsible for conducting backlog refinement as needed to better understand the Product Backlog items and the work in scope; the team does it together with the Product Owner. The team may (but does not have to) ask for support from the Scrum Master in organizing and helping in finding efficient ways of conducting the refinement. The Backlog Refinement includes issues from ServiceNow which have been previously evaluated to become part of the forecasted future Sprint Backlog.
Sprint Planning	R	Developers are responsible for conducting the Sprint Planning together with the Product Owner, in evaluating the work forecast and how-to for the upcoming sprint. The team may ask the Scrum Master for support in facilitating the event.
Daily Scrum	R	Developers are responsible for conducting the Daily Scrum, as a key inspect and adapt meeting. Developers may invite others to join the meeting if it helps them and does not disrupt the meeting itself.
Sprint Review	R	Developers are responsible for conducting the Sprint Review, with focus on briefly discussing the sprint progress (successes, problems and their resolutions), presenting a “Demo” of the work meeting the Definition of Done and answering questions. The team may ask the Scrum Master for support in facilitating the event.
Sprint Retrospective	R	Developers are responsible for conducting the Sprint Retrospective. As part of the Scrum Team, they should inspect the completed Sprint, identify action items and create an improvement plan for the next sprint. The team may ask the Scrum Master for support in facilitating the event.

- i** Responsible | Does the work to complete the task - at least 1 team member per task
 Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
 Consulted | Provides input to the work based on own expertise - no limit of team members
 Informed | Needs to be kept in the loop - no limit of team members

3.2.4 Kanban

For Kanban teams, Developers are accountable for delivering value, such as professional services, creative endeavors, and the design of both physical and software products by following Kanban practices :

- Visualization of the Workflow
- Limiting Work in Progress (WIP)
- Active management of work items in progress (Manage Flow)
- Make Policies Explicit (a process and flow policies like WIP limits, capacity allocation, Definition of Done)
- Implement Feedback loops (7 feedback opportunities/ optional Kanban meetings)
- Improve Collaboratively, Exolve Experimentally. Inspecting and adapting the team's Definition of Workflow

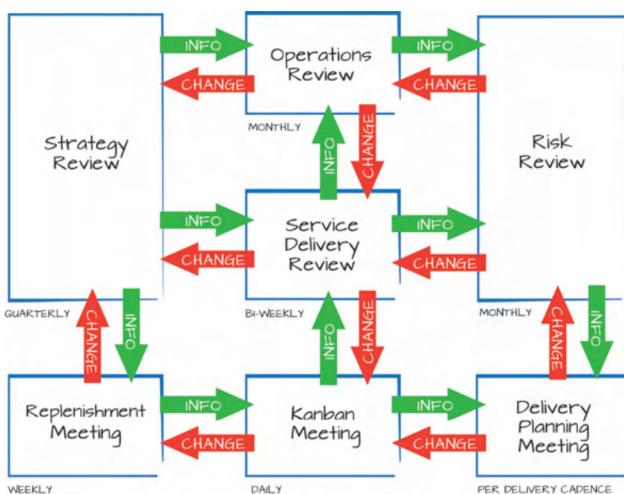
3.2.5 RACI matrix for optional Kanban meetings and sessions

All Meetings in Kanban are optional and should be decided by the team which should be used. However, they might be valuable from the "Implementing Feedback loops" perspective. Kanban defines seven specific feedback opportunities or cadences. Cadences are the cyclical meetings and reviews that drive evolutionary change and effective service delivery. "Cadence" may also refer to the time period between reviews—one workday or one month, for example: Choosing the right cadence is context-dependent and it is crucial to good outcomes. Too-frequent reviews may compel changing things before seeing the effect of previous changes, but if they are not frequent enough, poor performance may persist longer than necessary.

Replenishment Meeting	R	This meeting is for moving items over the commitment point (and into the system) and to oversee the preparation of options for future selection.
The Kanban Meeting	R	This is the (usually) daily coordination, self-organization, and planning review for those collaborating to deliver the service. It often uses a "stand-up" format to encourage a short, energetic meeting with the focus on completing work items and unblocking issues.
Strategy Review	R	This is for the selection of the services to be provided and to define for this set of services the concept of "fit for purpose"; also for sensing how the external environment is changing in order to provide direction to the services.
Operations Review	R	This is to understand the balance between and across services, deploying resources to maximize the delivery of value-aligned with customers' expectations.
Risk Review	R	This review is to understand and respond to the risks to effective delivery of services; for example, through blocker clustering
Service Delivery Review	R	This is to examine and improve the effectiveness of a service (this and subsequent cadences apply to a single service).
Delivery Planning Meeting	R	This is to monitor and plan deliveries to customers.

- ① Responsible | Does the work to complete the task - at least 1 team member per task
- Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
- Consulted | Provides input to the work based on own expertise - no limit of team members
- Informed | Needs to be kept in the loop - no limit of team members

Implementing the seven cadences does not imply adding seven new meetings to an organization's overhead, although the Replenishment and Kanban Meetings are considered a baseline in nearly all Kanban implementations.



(A set of cadences showing feedback loops infographic - <https://kanban.university/>)

3.2.6 Small Scale Scrum

The Three-People Team is comprised of the development team. The development team is expected to be involved in gathering and clarifying business requirements, doing development work, performing quality testing, and releasing software to the customer

3.2.7 The Sprint Review, Sprint Retrospective, and Sprint Planning may be combined into a single meeting which we term the Sprint Termination, lasting approximately 90 minutes. These meetings are concise, structured (thanks to advance preparation), and must not include any unnecessary/unrelated discussions.

Backlog refinement	R	Developers are responsible for conducting backlog refinement as needed to better understand the Product Backlog items and the work in scope; the team does it together with the Product Owner. The team may (but does not have to) ask for support from the Scrum Master in organizing and helping in finding efficient ways of conducting the refinement. The Backlog Refinement includes issues from ServiceNow which have been previously evaluated to become part of the forecasted future Sprint Backlog.
Sprint Planning	R	The Sprint Planning meeting is time-boxed (i.e., set in advance for a specified amount of time) and focused on planning work for the upcoming Sprint. The meeting is run by the development team and advance planning is required to keep it concise. Sprint Planning is value-based. At this point, requirements that will be worked on in the upcoming Sprint should be clear and contain approved acceptance criteria. Capacity, typically measured as velocity, is not used; instead, the team has to take an educated guess. Velocity only emerges after three or more Sprints, which is usually after the Small Scale Scrum projects are finished
Daily Scrum	R	Developers are responsible for conducting the Daily Scrum, as a key inspect and adapt meeting. Developers may invite others to join the meeting if it helps them and does not disrupt the meeting itself.
Sprint Review	R	The meeting is organized and run by the development team and attended by the customer or customer team. The Sprint Review is time-boxed and contains demonstrations of Sprint work and a short outline of work completed/not completed, bugs raised, and known and/or descoped issues (if any). Customer feedback is gathered at the end of the demonstration and incorporated into future Sprints. Very little (if any) preparation is required to keep the meeting short and structured.
Sprint Retrospective	R	The Sprint Retrospective meeting is organized and run by the development team and may be attended by the customer and/or customer team. The Sprint Retrospective is time-boxed and requires no advance preparation. Due to the small size of the development team and the Sprint builds (with a smaller number of features delivered), this meeting is relatively short. The Sprint Retrospective takes place after the Sprint Review and before the next Sprint Planning meeting
Proof of Concept/Demo	R	The Proof of Concept/Demo is a realization of the small work completed in the Sprint to showcase progress in development. Any deviations or incomplete work are discussed during the POC/demo.
First/Final Release	R	The First/Final Release is the project's end result. The development team runs the tests, verifies the completed work, confirms fixes, and signs off on the release build.

- ⓘ Responsible | Does the work to complete the task - at least 1 team member per task
- Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
- Consulted | Provides input to the work based on own expertise - no limit of team members
- Informed | Needs to be kept in the loop - no limit of team members

3.2.8 3.4.1 - The role of a Business Analyst / UX Designer (part of Developers)

NGA 2.0

In NGA 2.0 can be selected the best option for the team:

- EG Scrum
- Kanban
- Small Scale Scrum

And also we have gathered here best practices for teams who are struggling with:

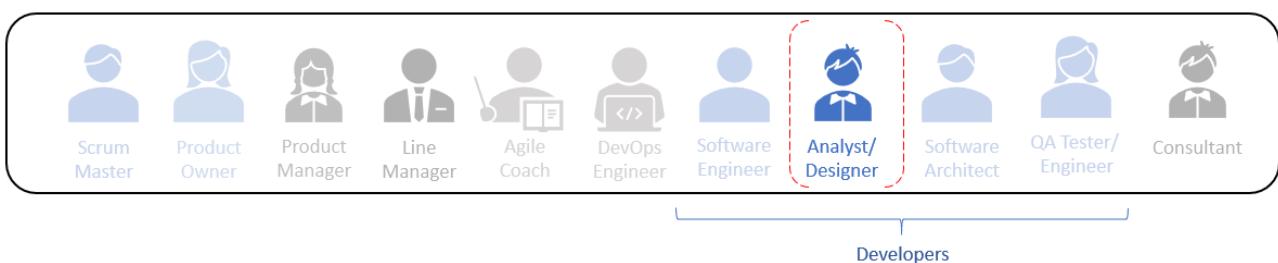
- Consultancy Teams
- High Granularity Teams

The above approaches will help Agile Coaches and Business Units to fill out their needs in the Agile way of working.

Independent of the team setup, a Business Analyst or a UX Designer is in practice a member of a Scrum Team with equal rights and responsibilities as any other team member (Software Engineer, QA Engineer). Business Analysts and UX Designers should contribute to the overall work being done with aim of delivering a releasable Increment of "Done" product at the end of each Sprint.

While the team placement of analysts and designers may not be directly within the Scrum Team, depending on the circumstances (they may be shared across more than 1 team), their accountability for the end product is equal of that of the Software Engineers who they support.

Dariusz Szlek, EG Agile Coach



IMPORTANT:

Business Analyst and UX designer roles, together with Software Engineers are defined as Developers in Scrum. We emphasize these two roles (BA & UX) separately in Agile Playbook due to their responsibilities

which differ from those of a Software Engineer. All of the mentioned roles though contribute to the final product increment.

3.2.8.1 Roles and duties of a Business Analyst / UX Designer

1. Refining Product Backlog items and supporting Software Engineers in decomposition/estimation
 - a. Learning the goal and value of Product Backlog items
 - b. Providing feedback in forms of functional analysis, specification, visual designs and additional insight, which may influence the change of the Product Backlog (items)
 - c. Supporting Software Engineers in the decomposition of epics into smaller epics or user stories (more in [5.3 - The Product Backlog \(see page 160\)](#) and [4.2 - The Backlog Refinement \(see page 113\)](#)), as well as user stories into sub-tasks (more in [4.3 - The Sprint Planning \(see page 117\)](#))
 - d. Supporting the Software Engineers in estimating epics high-level and user stories, by providing relevant input and knowledge about the Product Backlog items ([5.2 - Refining and estimating an initial Product Backlog \(see page 152\)](#))
2. Focusing on the delivery of a potentially releasable product increment
 - a. Working on sprint backlog items with respect to their complexity and priority
 - b. Verifying the validity of the sprint goal on a daily basis
 - c. Supporting Software Engineers in delivery of a releasable product increment at the end of the sprint
 - d. Gathering requirements, designing, implementing and testing solutions as part of the product
 - e. Supporting the Product Owner in building a proper and essential Product Backlog, as well as filling the content of Product Backlog items with valuable, functional and visual insight
3. Enforcing transparency through the use of EG agile software development tooling stack
 - a. Using Jira for development workflow management and specifically scrum boards, as well as dashboards for transparent work progress visualization
 - b. Using Confluence for documentation
 - c. Using BitBucket as the code repository
 - d. Keeping the state of work progress and documentation up to date in the mentioned tools
4. Participating in all Scrum Events as needed
5. Using metrics and artefacts as intended
 - a. Owning the sprint backlog and working on the product backlog together with Software Engineers
 - b. Checking and confirming the Definition of Ready (more in [5.5 - The Definition of Ready \(see page 176\)](#)) as well as the Definition of Done (more in [5.6 - The Definition of Done \(see page 180\)](#)) for all Backlog items
 - c. Keeping team information, metrics, rules and agreements up to date (more in [5.8 - Team page \(see page 211\)](#))
 - d. Using velocity and capacity (more in [5.7 - Using team metrics \(see page 184\)](#)) as input in evaluating the sprint and release plans
6. Supporting self-management and continuous improvement within the Scrum Team

- a. Learning from Agile Coaches, Scrum Master and this playbook about proper implementation of Scrum in EG and following the guidelines
- b. Taking ownership of Scrum Events, sessions, metrics and artefacts together with Software Engineers when external support is unnecessary
- c. Providing feedback to the Scrum Master/Agile Coach and suggesting improvements in all areas of the Scrum Team operation
- d. Checking own work for quality, consistency and coherence with the sprint/release goal
- e. Identifying and implementing improvements in tooling and processes to improve the work of the Scrum Team

3.2.8.2 The collaboration of a Business Analyst / UX Designer with Software Engineers

- Daily communication and cooperation in the scope of current Sprint Backlog items and Backlog items projected for the nearest upcoming sprints
- Active support in refinement, estimation and verification of completed and planned work together with Software Engineers, as deemed necessary by the team
- Providing valuable feedback, insight, documentation, designs and clarification of Product Backlog Items, which are in progress or planned, to the required level of detail
- Contributing to the well-being, self-management and continuous improvement of the entire Scrum Team

3.2.8.3 The collaboration of a Business Analyst / UX Designer with the Product Owner

- Support in specification and clarification of functional/design aspects of Product Backlog items, by providing additional insight on top of fundamental understanding of its purpose
- Support in testing and verification of the developed solution, providing feedback and improvement opportunities
- Contributing to the best possible understanding by Software Engineers of the purpose and goal of building the solution/feature desired by the Product Owner

3.2.8.4 RACI matrix for EG Scrum events and sessions

Backlog refinement	R	The Business Analyst and UX Designer are responsible for participating in Software Engineer's backlog refinement as needed, to help better understand the Product Backlog items and the work in scope from the functional, analytical and design perspective; the team does it together with the Product Owner. The team may (but does not have to) ask for support from the Scrum Master in organizing and helping in finding efficient ways of conducting the refinement.
---------------------------	---	---

Sprint Planning	R	The Business Analyst and UX Designer are responsible for conducting the Sprint Planning together with Software Engineers and the Product Owner, in evaluating the work forecast and how-to for the upcoming sprint. They provide all the necessary insight and support to Software Engineers needed to establish the forecast. The team may ask the Scrum Master for support in facilitating the event.
Daily Scrum	R	The Business Analyst and UX Designer are responsible for participating in the Daily Scrum together with their Software Engineers, as a key inspect and adapt meeting. Software Engineers may decide on further regular participation of analysts and designers depending on the added value if it helps them and does not disrupt the meeting itself.
Sprint Review	R	The Business Analyst and UX Designer are responsible for participating in the Sprint Review, with focus on briefly discussing the sprint progress (successes, problems and their resolutions), supporting the “Demo” of the work meeting the Definition of Done (presented by the Developers), adding additional insight into the designed processes/visuals and answering related questions. The team may ask the Scrum Master for support in facilitating the event.
Sprint Retrospective	R	The Business Analyst and UX Designer are responsible for participating in the Sprint Retrospective. As part of the Scrum Team, they should inspect the completed Sprint, identify action items and create an improvement plan for the next sprint. The team may ask the Scrum Master for support in facilitating the event.

- i** Responsible | Does the work to complete the task - at least 1 team member per task
Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
Consulted | Provides input to the work based on own expertise - no limit of team members
Informed | Needs to be kept in the loop - no limit of team members

3.2.9 3.4.2 - The role of SW Architect

NGA 2.0

In NGA 2.0 can be selected the best option for the team:

- EG Scrum
- Kanban
- Small Scale Scrum

And we have gathered here best practices for teams who are struggling with:

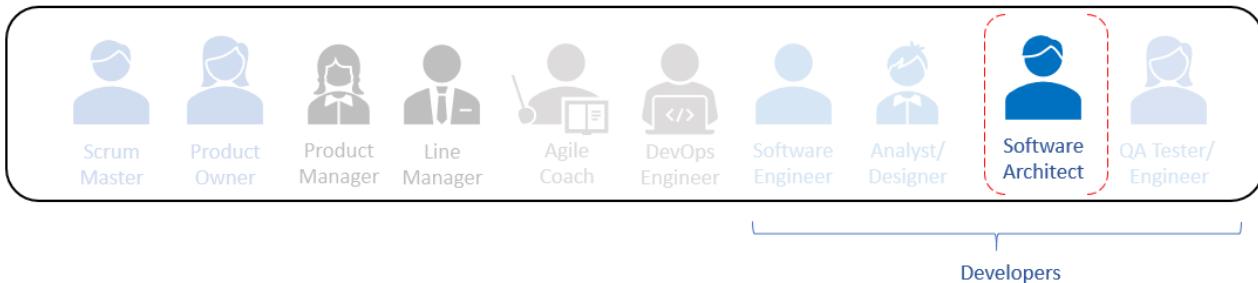
- Consultancy Teams
- High Granularity Teams

The above approaches will help Agile Coaches and Business Units to fill out their needs in the Agile way of working.

SW Architect is a Developer in the Scrum Team, that facilitates discussion in regards to technical solutions for building a system or specific functionalities. SW Architect is not a single point of decision. He/She is accountable for crafting together with the Developers, the best technical solution for achieving required system or functionality qualities.

The SW Architect is accountable for:

- Active refinement of backlog items with special attention to non-functional requirements (availability, modifiability, performance, scalability, testability, usability)
- Facilitate discussion with Developers and joint decision-making in regards to applying best available technical solution and best implementation methods
- Facilitate discussion with Developers and crafting designs of logic for the system or specific functionality
- Inspire to crafting and maintaining technical principles, tradeoffs, and patterns
- Inspire to architecture adaptation in order to meet changing requirements and reduce technological debt
- Adjusting architectural items into the backlog (enablers)



IMPORTANT:

SW Architect in EG is more a competency than a role. SW Architect is still a Scrum Team Developer, committed to creating any aspect of a usable Increment each Sprint. Addition here is engaging technically oriented people to secure system qualities.

3.2.9.1 SW Architects same as Developers should strive for being:

- Self-managing - the team needs to evaluate and decide on their own how to work together on delivering the potentially releasable product increment every sprint. The team may seek council outside of the team if necessary, to obtain specific input required to progress towards the sprint goal (such as UX Designer, DevOps Engineer, Agile Coach, etc.). However, such external dependencies should always be minimized by fostering competencies required to deliver value by the team itself. No one should instruct the team on how to execute their tasks otherwise.
- Cross-functional - the team needs to possess all known skills, competencies, and tools with respect to the product backlog which is known to be processed at a given moment. The team may be composed of specialists in particular domains, who as a group contribute to building the product

cross-functionally, however, the team should aim to distribute such skills and knowledge throughout the team to avoid bottlenecks. Should a need for new competencies arise, the team should plan to gain the appropriate tools and skills in the scope of their sprint, in order to complete the sprint backlog items.

- Flat-structured - the team does not recognize any titles or statuses within it. All members of the team are equal regardless of the executed tasks and all members share equal accountability for the end result. The team's size should fit between 3 and 8 members including all roles necessary for a cross-functional delivery of the product increment.

3.2.9.2 Responsibilities and duties of an SW Architect

3.2.9.3 SW Architects are a Developers from Scrum setup perspective, so they should fulfill all responsibilities related to the Developer role:

1. Refining, breaking down, and estimating Product Backlog items
 - a. Learning the goal and value of Product Backlog items
 - b. Providing feedback and proposing changes to the Product Backlog items and the Product Backlog
 - c. Decomposing epics into smaller epics or user stories (more in [5.3 - The Product Backlog \(see page 160\)](#) and [4.2 - The Backlog Refinement \(see page 113\)](#)) and user stories into sub-tasks (more in [4.3 - The Sprint Planning \(see page 117\)](#))
 - d. Designing technical architecture. Discussing and joint decision making regarding used technologies, solutions, based on quality goals (non-functional acceptance criteria like: portability, maintainability, security, reliability, functional suitability, performance efficiency, compatibility, usability). More about the Agile approach to SW Architecture [scrum.org reference⁸](#)
 - e. Estimating epics high-level and user stories, using Story Points or other agreed methods (more in [5.2 - Refining and estimating an initial Product Backlog \(see page 152\)](#))
 - f. Handling incoming issues from ServiceNow in terms of initial evaluation to support the Product Owner with input to the decision regarding placement of the issue in the Product Backlog
 - g. Communicating with ServiceNow support to acquire any necessary data, that is not in the scope of the teams' competencies, for handling an incoming ServiceNow support issue and further assessment
2. Working on delivery of a potentially releasable product increment
 - a. Working on sprint backlog items with respect to their complexity and priority
 - b. Verifying the validity of the sprint goal on a daily basis
 - c. Striving for delivery of a releasable product increment at the end of the sprint

⁸ <https://www.scrum.org/pathway/software-developer-learning-path/developing-and-delivering-products-professionally/emergent-software-development>

- d. Designing, implementing, and Testing (more in [Quality Management at EG⁹](#)), as inseparable elements of the product development and value delivery
 - e. Release Notes is a document containing information about new features, changes in the products, or product updates. During the release of a usable product or part of it, developers have to prepare release notes. Full instruction on how to do them is in chapter [7.3 - Versioning and release notes \(see page 323\)](#)
3. Enforcing transparency through the use of EG agile software development tooling stack
- a. Using Jira for development workflow management and specifically scrum boards, as well as dashboards for transparent work progress visualization
 - b. Using Confluence for documentation
 - c. Using Bitbucket as the code repository
 - d. Keeping the state of work progress and documentation up to date in the mentioned tools
4. Participating in all Scrum Events
5. Enforcing inspection and adaptation, as part of continuous improvement
- a. Seeking solutions to malfunctioning processes within the team and implementing them
 - b. Checking own work for quality, consistency, and coherence with the sprint/release goal
 - c. Looking for improvements in tooling and practices to improve the work of the team and implementing them
 - d. Providing feedback to the Scrum Master and suggesting improvements in any other area of the team
6. Striving for increased self-management and autonomy
- a. Learning proper implementation of Scrum in EG and following the guidelines
 - b. Aiming for achieving an autonomous team setup with fully dedicated team members
 - c. Taking ownership of Scrum Events, sessions, metrics, and artefacts when external support is unnecessary
7. Using metrics and artifacts as intended
- a. Owning the sprint backlog and working on the product backlog
 - b. Checking and confirming the Definition of Ready (more in [5.5 - The Definition of Ready \(see page 176\)](#)) as well as the Definition of Done (more in [5.6 - The Definition of Done \(see page 180\)](#)) for Backlog items
 - c. Keeping team information, metrics, rules, and agreements up to date (more in [5.8 - Team page \(see page 211\)](#))
 - d. Using velocity and capacity (more in [5.7 - Using team metrics \(see page 184\)](#)) as input in evaluating the sprint and release plans

⁹ <https://confluence.eg.dk/display/NG/Quality+Management+at+EG>

3.2.9.4 RACI matrix for EG Scrum events and sessions

Backlog refinement	R	Developers are responsible for conducting backlog refinement as needed to better understand the Product Backlog items and the work in scope; the team does it together with the Product Owner. The team may (but does not have to) ask for support from the Scrum Master in organizing and helping in finding efficient ways of conducting the refinement. The Backlog Refinement includes issues from ServiceNow which have been previously evaluated to become part of the forecasted future Sprint Backlog.
Sprint Planning	R	Developers are responsible for conducting the Sprint Planning together with the Product Owner, in evaluating the work forecast and how-to for the upcoming sprint. The team may ask the Scrum Master for support in facilitating the event.
Daily Scrum	R	Developers are responsible for conducting the Daily Scrum, as a key inspect and adapt meeting. Developers may invite others to join the meeting if it helps them and does not disrupt the meeting itself.
Sprint Review	R	Developers are responsible for conducting the Sprint Review, with a focus on briefly discussing the sprint progress (successes, problems, and their resolutions), presenting a "Demo" of the work meeting the Definition of Done, and answering questions. The team may ask the Scrum Master for support in facilitating the event.
Sprint Retrospective	R	Developers are responsible for conducting the Sprint Retrospective. As part of the Scrum Team, they should inspect the completed Sprint, identify action items and create an improvement plan for the next sprint. The team may ask the Scrum Master for support in facilitating the event.

Responsible | Does the work to complete the task - at least 1 team member per task
Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
Consulted | Provides input to the work based on own expertise - no limit of team members
Informed | Needs to be kept in the loop - no limit of team members

3.2.9.5 Kanban

SW Architects holds a Developer role also for Kanban teams, so he/she should fulfill all responsibilities related to the Developer role. For Kanban teams, Developers are accountable for delivering value, such as professional services, creative endeavors, and the design of both physical and software products by following Kanban practices:

- Visualization of the Workflow
- Limiting Work in Progress (WIP)
- Active management of work items in progress (Manage Flow)
- Make Policies Explicit (a process and flow policies like WIP limits, capacity allocation, Definition of Done)
- Implement Feedback loops (7 feedback opportunities/ optional Kanban meetings)
- Improve Collaboratively, Experiments Experimentally. Inspecting and adapting the team's Definition of Workflow

3.2.9.6 RACI matrix for optional Kanban meetings and sessions

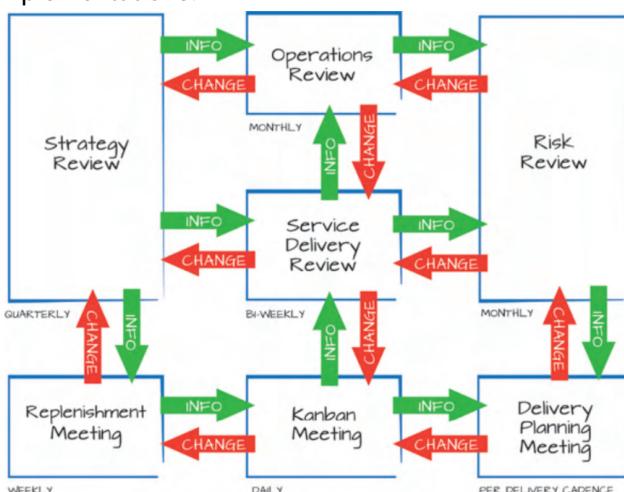
All Meetings in Kanban are optional and should be decided by the team which should be used. However, they might be valuable from the "Implementing Feedback loops" perspective. Kanban defines seven specific feedback opportunities or cadences. Cadences are the cyclical meetings and reviews that drive evolutionary change and effective service delivery. "Cadence" may also refer to the time period between reviews—one workday or one month, for example: Choosing the right cadence is context-dependent and it is crucial to good outcomes. Too-frequent reviews may compel changing things before seeing the effect of previous changes, but if they are not frequent enough, poor performance may persist longer than necessary.

Replenishment Meeting	R	This meeting is for moving items over the commitment point (and into the system) and overseeing the preparation of options for future selection.
The Kanban Meeting	R	This is the (usually) daily coordination, self-organization, and planning review for those collaborating to deliver the service. It often uses a "stand-up" format to encourage a short, energetic meeting with the focus on completing work items and unblocking issues.
Strategy Review	R	This is for the selection of the services to be provided and to define for this set of services the concept of "fit for purpose"; also, for sensing how the external environment is changing in order to provide direction to the services.

Operations Review	R	This is to understand the balance between and across services, deploying resources to maximize the delivery of value aligned with customers' expectations.
Risk Review	R	This review is to understand and respond to the risks to the effective delivery of services; for example, through blocker clustering
Service Delivery Review	R	This is to examine and improve the effectiveness of a service (this and subsequent cadences apply to a single service).
Delivery Planning Meeting	R	This is to monitor and plan deliveries to customers.

Responsible | Does the work to complete the task - at least 1 team member per task
Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
Consulted | Provides input to the work based on own expertise - no limit of team members
Informed | Needs to be kept in the loop - no limit of team members

Implementing the seven cadences does not imply adding seven new meetings to an organization's overhead, although the Replenishment and Kanban Meetings are considered a baseline in nearly all Kanban implementations.



(A set of cadences showing feedback loops infographic - <https://kanban.university/>)

3.2.9.7 Small Scale Scrum

The Three-People Team is comprised of the development team. The development team is expected to be involved in gathering and clarifying business requirements, doing development work, performing quality testing, and releasing software to the customer. SW Architecture is a competency that the development team should have and use to secure system qualities.

3.2.9.8 The Sprint Review, Sprint Retrospective, and Sprint Planning may be combined into a single meeting which we term the Sprint Termination, lasting approximately 90 minutes. These meetings are concise, structured (thanks to advance preparation), and must not include any unnecessary/unrelated discussions.

Backlog refinement	R	Developers are responsible for conducting backlog refinement as needed to better understand the Product Backlog items and the work in scope; the team does it together with the Product Owner. The team may (but does not have to) ask for support from the Scrum Master in organizing and helping in finding efficient ways of conducting the refinement. The Backlog Refinement includes issues from ServiceNow which have been previously evaluated to become part of the forecasted future Sprint Backlog.
Sprint Planning	R	The Sprint Planning meeting is time-boxed (i.e., set in advance for a specified amount of time) and focused on planning work for the upcoming Sprint. The meeting is run by the development team and advance planning is required to keep it concise. Sprint Planning is value-based. At this point, requirements that will be worked on in the upcoming Sprint should be clear and contain approved acceptance criteria. Capacity, typically measured as velocity, is not used; instead, the team has to take an educated guess. Velocity only emerges after three or more Sprints, which is usually after the Small Scale Scrum projects are finished
Daily Scrum	R	Developers are responsible for conducting the Daily Scrum, as a key inspect and adapt meeting. Developers may invite others to join the meeting if it helps them and does not disrupt the meeting itself.
Sprint Review	R	The meeting is organized and run by the development team and attended by the customer or customer team. The Sprint Review is time-boxed and contains demonstrations of Sprint work and a short outline of work completed/not completed, bugs raised, and known and/or descoped issues (if any). Customer feedback is gathered at the end of the demonstration and incorporated into future Sprints. Very little (if any) preparation is required to keep the meeting short and structured.

Sprint Retrospective	R	The Sprint Retrospective meeting is organized and run by the development team and may be attended by the customer and/or customer team. The Sprint Retrospective is time-boxed and requires no advance preparation. Due to the small size of the development team and the Sprint builds (with a smaller number of features delivered), this meeting is relatively short. The Sprint Retrospective takes place after the Sprint Review and before the next Sprint Planning meeting
Proof of Concept/Demo	R	The Proof of Concept/Demo is a realization of the small work completed in the Sprint to showcase progress in development. Any deviations or incomplete work are discussed during the POC/demo.
First/Final Release	R	The First/Final Release is the project's end result. The development team runs the tests, verifies the completed work, confirms fixes, and signs off on the release build.

Responsible | Does the work to complete the task - at least 1 team member per task
Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
Consulted | Provides input to the work based on own expertise - no limit of team members
Informed | Needs to be kept in the loop - no limit of team members

3.2.10 3.4.3 - The role of QA Tester / Engineer

NGA 2.0

In NGA 2.0 can be selected the best option for the team:

- EG Scrum
- Kanban
- Small Scale Scrum

And we have gathered here best practices for teams who are struggling with:

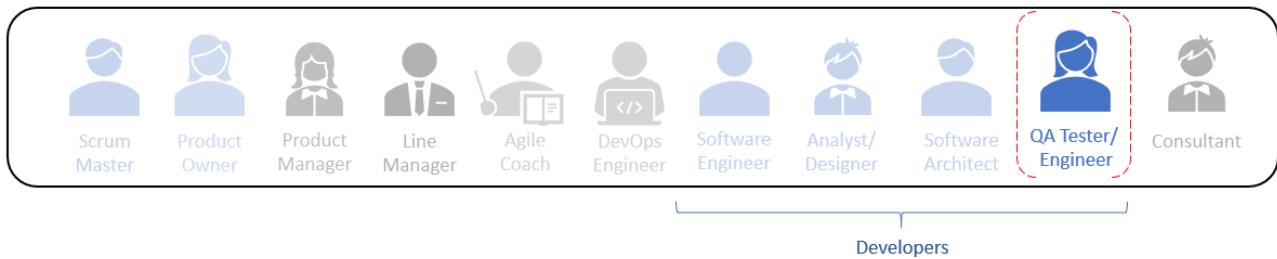
- Consultancy Teams
- High Granularity Teams

The above approaches will help Agile Coaches and Business Units to fill out their needs in the Agile way of working.

QA Tester / Engineer is a Developer in the Scrum Team, committed to creating any aspect of a usable Increment each Sprint. He / She is accountable for crafting the required system or functionality with the team, emphasizing built-in quality during the software development process:

- Active refinement of backlog items with special attention to testability, functional and non-functional requirements (e.g. security, performance)
- Creating a plan for the Sprint, the Sprint Backlog;

- Instilling quality by adhering to a Definition of Done;
- Adapting their plan each day toward the Sprint Goal; and,
- Holding each other accountable as professionals.



IMPORTANT:

QA Tester / Engineer in EG is more competency than a role. QA Tester / Engineer is still a Scrum Team Developer, committed to creating any aspect of a usable Increment each Sprint. In addition, there is special attention to ensuring Built-in Quality during the development process and finally to the product.

3.2.10.1 **QA Testers / Engineers** same as Developers should strive for being:

- Self-managing - the team needs to evaluate and decide on their own how to work together on delivering the potentially releasable product increment every Sprint. If necessary, the team may seek a council outside of the team to obtain specific input required to progress towards the sprint goal (UX Designer, DevOps Engineer, Agile Coach, etc.). However, such external dependencies should always be minimized by fostering competencies required to deliver value by the team itself. No one should instruct the team on how to execute their tasks otherwise.
- Cross-functional - the team needs to possess all known skills, competencies, and tools regarding the product backlog that is known to be processed at a given moment. The team may be composed of specialists in particular domains who contribute to building the product cross-functionally; however, the team should aim to distribute such skills and knowledge to avoid bottlenecks. Should a need for new competencies arise, the team should plan to gain the appropriate tools and skills in the scope of their Sprint to complete the sprint backlog items.
- Flat-structured - the team does not recognize any titles or statuses within it. All team members are equal regardless of the executed tasks, and all members share equal accountability for the end result. The team's size should fit between 3 and 8 members, including all roles necessary for a cross-functional delivery of the product increment.

3.2.10.2 Responsibilities and duties of a QA Tester / Engineer

3.2.10.3 **QA Testers / Engineers are a Developers from Scrum setup perspective, so they should fulfill all responsibilities related to the Developer role:**

1. Refining, breaking down, and estimating Product Backlog items

- a. Learning the goal and value of Product Backlog items
 - b. Providing feedback and proposing changes to the Product Backlog items and the Product Backlog
 - c. Decomposing epics into smaller epics or user stories (more in [5.3 - The Product Backlog \(see page 160\)](#) and [4.2 - The Backlog Refinement \(see page 113\)](#)) and user stories into sub-tasks (more in [4.3 - The Sprint Planning \(see page 117\)](#))
 - d. Developing and designing test cases and /or test automation scripts based on the [test basis¹⁰](#) to cover goals defined in the acceptance criteria
 - e. Estimating epics high-level and user stories, using Story Points or other agreed methods (more in [5.2 - Refining and estimating an initial Product Backlog \(see page 152\)](#))
 - f. Handling incoming issues from ServiceNow in terms of initial evaluation to support the Product Owner with input to the decision regarding placement of the issue in the Product Backlog
 - g. Communicating with ServiceNow support to acquire any necessary data, that is not in the scope of the teams' competencies, for handling an incoming ServiceNow support issue and further assessment
2. Working on delivery of a potentially releasable product increment
- a. Working on sprint backlog items with respect to their complexity and priority
 - b. Verifying the validity of the sprint goal on a daily basis
 - c. Striving for delivery of a releasable product increment at the end of the sprint
 - d. Designing, implementing, and Testing (more in [Quality Management at EG¹¹](#)), as inseparable elements of the product development and value delivery
 - e. Executing tests and ensuring efficient reporting of detected defects
 - f. Release Notes is a document containing information about new features, changes in the products, or product updates. During the release of a usable product or part of it, developers have to prepare release notes. Full instruction on how to do them is in chapter [7.3 - Versioning and release notes \(see page 323\)](#)
3. Enforcing transparency through the use of EG agile software development tooling stack
- a. Using Jira for development workflow management and specifically scrum boards, as well as dashboards for transparent work progress visualization
 - b. Using Zephyr Scale for test management, including test: planning, estimation, designing, execution and monitoring, and reporting but also for full test traceability
 - c. Using Confluence for documentation
 - d. Using Bitbucket as the code repository
 - e. Keeping the state of work progress and documentation up to date in the mentioned tools
4. Participating in all Scrum Events
5. Enforcing inspection and adaptation, as part of continuous improvement
- a. Seeking solutions to malfunctioning processes within the team and implementing them
 - b. Checking own work for quality, consistency, and coherence with the sprint/release goal

¹⁰ <https://glossary.istqb.org/en/term/test-basis-3>

¹¹ <https://confluence.eg.dk/display/NG/Quality+Management+at+EG>

- c. Contributing to Community of Practice for Testing and QA to continuously influence the growth and improvements of the entire organization from the quality perspective and to share the ideas with other BUs
 - d. Looking for improvements in tooling and practices to improve the work of the team and implementing them
 - e. Providing feedback to the Scrum Master and suggesting improvements in any other area of the team
6. Striving for increased self-management and autonomy
- a. Learning proper implementation of Scrum in EG and following the guidelines
 - b. Aiming for achieving an autonomous team setup with fully dedicated team members
 - c. Taking ownership of Scrum Events, sessions, metrics, and artifacts when external support is unnecessary
7. Using metrics and artifacts as intended
- a. Owning the sprint backlog and working on the product backlog
 - b. Checking and confirming the Definition of Ready (more in [5.5 - The Definition of Ready \(see page 176\)](#)) as well as the Definition of Done (more in [5.6 - The Definition of Done \(see page 180\)](#)) for Backlog items
 - c. Keeping team information, metrics, rules, and agreements up to date (more in [5.8 - Team page \(see page 211\)](#))
 - d. Using velocity and capacity (more in [5.7 - Using team metrics \(see page 184\)](#)) as input in evaluating the sprint and release plans

3.2.10.4 RACI matrix for EG Scrum events and sessions

Backlog refinement	R	Developers are responsible for conducting backlog refinement as needed to better understand the Product Backlog items and the work in scope; the team does it together with the Product Owner. The team may (but does not have to) ask for support from the Scrum Master in organizing and helping in finding efficient ways of conducting the refinement. The Backlog Refinement includes issues from ServiceNow which have been previously evaluated to become part of the forecasted future Sprint Backlog.
Sprint Planning	R	Developers are responsible for conducting the Sprint Planning together with the Product Owner, in evaluating the work forecast and how-to for the upcoming sprint. The team may ask the Scrum Master for support in facilitating the event.

Daily Scrum	R	Developers are responsible for conducting the Daily Scrum, as a key inspect and adapt meeting. Developers may invite others to join the meeting if it helps them and does not disrupt the meeting itself.
Sprint Review	R	Developers are responsible for conducting the Sprint Review, with a focus on briefly discussing the sprint progress (successes, problems, and their resolutions), presenting a "Demo" of the work meeting the Definition of Done, and answering questions. The team may ask the Scrum Master for support in facilitating the event.
Sprint Retrospective	R	Developers are responsible for conducting the Sprint Retrospective. As part of the Scrum Team, they should inspect the completed Sprint, identify action items and create an improvement plan for the next sprint. The team may ask the Scrum Master for support in facilitating the event.

Responsible | Does the work to complete the task - at least 1 team member per task
Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
Consulted | Provides input to the work based on own expertise - no limit of team members
Informed | Needs to be kept in the loop - no limit of team members

3.2.10.5 Kanban

QA Tester / Engineer holds a Developer role also for Kanban teams, so he/she should fulfill all responsibilities related to the Developer role. For Kanban teams, Developers are accountable for delivering value, such as professional services, creative endeavors, and the design of both physical and software products by following Kanban practices:

- Visualization of the Workflow
- Limiting Work in Progress (WIP)
- Active management of work items in progress (Manage Flow)
- Make Policies Explicit (a process and flow policies like WIP limits, capacity allocation, Definition of Done)
- Implement Feedback loops (7 feedback opportunities/ optional Kanban meetings)
- Improve Collaboratively, Exolve Experimentally. Inspecting and adapting the team's Definition of Workflow

3.2.10.6 RACI matrix for optional Kanban meetings and sessions

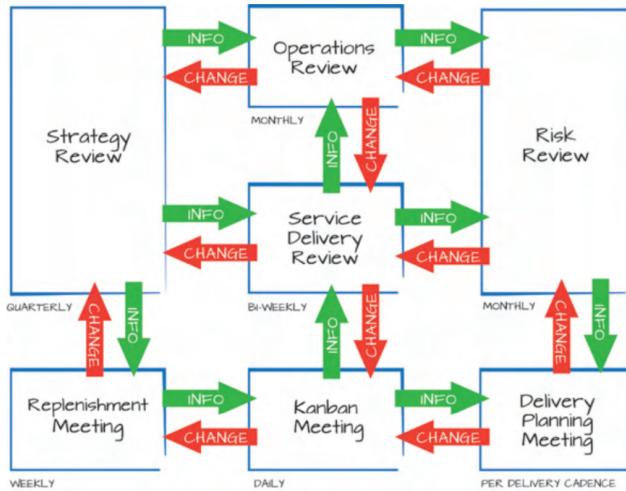
All Meetings in Kanban are optional and should be decided by the team which should be used. However, they might be valuable from the "Implementing Feedback loops" perspective. Kanban defines seven specific feedback opportunities or cadences. Cadences are the cyclical meetings and reviews that drive evolutionary change and effective service delivery. "Cadence" may also refer to the time period between reviews—one workday or one month, for example: Choosing the right cadence is context-dependent and it is crucial to

good outcomes. Too-frequent reviews may compel changing things before seeing the effect of previous changes, but if they are not frequent enough, poor performance may persist longer than necessary.

Replenishment Meeting	R	This meeting is for moving items over the commitment point (and into the system) and overseeing the preparation of options for future selection.
The Kanban Meeting	R	This is the (usually) daily coordination, self-organization, and planning review for those collaborating to deliver the service. It often uses a "stand-up" format to encourage a short, energetic meeting with the focus on completing work items and unblocking issues.
Strategy Review	R	This is for the selection of the services to be provided and to define for this set of services the concept of "fit for purpose"; also, for sensing how the external environment is changing in order to provide direction to the services.
Operations Review	R	This is to understand the balance between and across services, deploying resources to maximize the delivery of value aligned with customers' expectations.
Risk Review	R	This review is to understand and respond to the risks to the effective delivery of services; for example, through blocker clustering
Service Delivery Review	R	This is to examine and improve the effectiveness of a service (this and subsequent cadences apply to a single service).
Delivery Planning Meeting	R	This is to monitor and plan deliveries to customers.

Responsible | Does the work to complete the task - at least 1 team member per task
Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
Consulted | Provides input to the work based on own expertise - no limit of team members
Informed | Needs to be kept in the loop - no limit of team members

Implementing the seven cadences does not imply adding seven new meetings to an organization's overhead, although the Replenishment and Kanban Meetings are considered a baseline in nearly all Kanban implementations.



(A set of cadences showing feedback loops infographic - <https://kanban.university/>)

3.2.10.7 Small Scale Scrum

The Three-People Team is comprised of the development team. The development team is expected to be involved in gathering and clarifying business requirements, doing development work, performing quality testing, and releasing software to the customer. SW Quality Assurance is a competency that the development team should have and use to secure system qualities.

3.2.10.8 The Sprint Review, Sprint Retrospective, and Sprint Planning may be combined into a single meeting which we term the Sprint Termination, lasting approximately 90 minutes. These meetings are concise, structured (thanks to advance preparation), and must not include any unnecessary/unrelated discussions.

Backlog refinement	R	Developers are responsible for conducting backlog refinement as needed to better understand the Product Backlog items and the work in scope; the team does it together with the Product Owner. The team may (but does not have to) ask for support from the Scrum Master in organizing and helping in finding efficient ways of conducting the refinement. The Backlog Refinement includes issues from ServiceNow which have been previously evaluated to become part of the forecasted future Sprint Backlog.
---------------------------	---	--

Sprint Planning	R	The Sprint Planning meeting is time-boxed (i.e., set in advance for a specified amount of time) and focused on planning work for the upcoming Sprint. The meeting is run by the development team and advance planning is required to keep it concise. Sprint Planning is value-based. At this point, requirements that will be worked on in the upcoming Sprint should be clear and contain approved acceptance criteria. Capacity, typically measured as velocity, is not used; instead, the team has to take an educated guess. Velocity only emerges after three or more Sprints, which is usually after the Small Scale Scrum projects are finished
Daily Scrum	R	Developers are responsible for conducting the Daily Scrum, as a key inspect and adapt meeting. Developers may invite others to join the meeting if it helps them and does not disrupt the meeting itself.
Sprint Review	R	The meeting is organized and run by the development team and attended by the customer or customer team. The Sprint Review is time-boxed and contains demonstrations of Sprint work and a short outline of work completed/not completed, bugs raised, and known and/or descoped issues (if any). Customer feedback is gathered at the end of the demonstration and incorporated into future Sprints. Very little (if any) preparation is required to keep the meeting short and structured.
Sprint Retrospective	R	The Sprint Retrospective meeting is organized and run by the development team and may be attended by the customer and/or customer team. The Sprint Retrospective is time-boxed and requires no advance preparation. Due to the small size of the development team and the Sprint builds (with a smaller number of features delivered), this meeting is relatively short. The Sprint Retrospective takes place after the Sprint Review and before the next Sprint Planning meeting
Proof of Concept/Demo	R	The Proof of Concept/Demo is a realization of the small work completed in the Sprint to showcase progress in development. Any deviations or incomplete work are discussed during the POC/demo.
First/Final Release	R	The First/Final Release is the project's end result. The development team runs the tests, verifies the completed work, confirms fixes, and signs off on the release build.

Responsible | Does the work to complete the task - at least 1 team member per task
Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
Consulted | Provides input to the work based on own expertise - no limit of team members
Informed | Needs to be kept in the loop - no limit of team members

3.3 3.3 - The role of a Product Owner and Product Manager

NGA 2.0

In NGA 2.0 can be selected the best option for the team:

- EG Scrum
- Kanban
- Small Scale Scrum

And also we have gathered here best practices for teams who are struggling with:

- Consultancy Teams
- High Granularity Teams

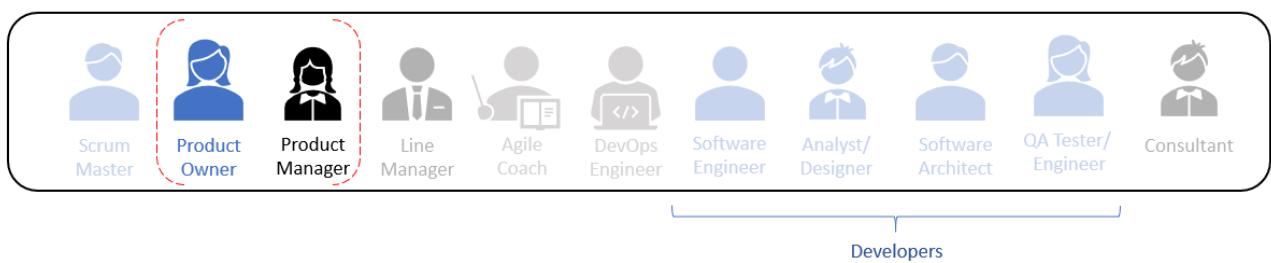
The above approaches will help Agile Coaches and Business Units to fill out their needs in the Agile way of working.

The Product Owner is accountable for maximizing the value of the product resulting from the work of the Scrum Team. How this is done may vary widely across organizations, Scrum Teams, and individuals.

*The Product Owner is also accountable for effective Product Backlog management.
 (...)*

The Product Owner is one person, not a committee. The Product Owner may represent the needs of many stakeholders in the Product Backlog. Those wanting to change the Product Backlog can do so by trying to convince the Product Owner.

The Scrum Guide, SCRUM.org



3.3.1 Responsibilities and duties of a Product Owner

1. Managing the Product Backlog

- a. Creating and updating Product Backlog items using clearly defined acceptance criteria, descriptions, compliant with the Definition of Ready (or as close as possible before the involvement of Developers)
 - b. Removing outdated or invaluable Product Backlog items
 - c. Prioritizing Product Backlog items with respect to business value, suggestions from the Developers (complexity, dependencies, redundancy) and target release plans
 - d. Ensuring proper marking and versioning of Product Backlog items in Jira and Confluence
 - e. Enforcing transparency of the Product Backlog and its progress through EG tools, such as Jira
 - f. Assessing ServiceNow tickets from support in terms of their value and priority as part of the Product Backlog scope
 - g. Deciding whether or not ServiceNow tickets from support are to be directly incorporated into the Product Backlog or revised against the Product roadmap with the Product Manager
2. Maximizing the value of the Product
 - a. Optimizing the value of the work performed by the Developers
 - b. Being the single source of contact and information for the Developers to address any issues regarding the product
 - c. Evaluating the current value of Product Backlog items with respect to changing conditions and information on a daily basis
 - d. Listening for input from the Developers to understand and acknowledge/react to the value/complexity ratio of the questionable Product Backlog items
 - e. Providing feedback and support to the Product Manager in terms of the product vision and roadmap
 3. Collaborating closely with the Developers
 - a. Clarifying any questions, issues, challenges or event impediments with the highest priority for the Developers in order to remove blockers and allow for progressing work
 - b. Resolving uncertainties, gaps, misinterpretations at any moment of the sprint for the Developers
 - c. Reviewing and providing feedback on the work of the Developers as soon as possible
 - d. Ensuring that the Developers understand the product vision, sprint goal and all associated aspects of the Product Backlog
 - e. Approving Product Backlog items made ready by the Developers in accordance to the Definition of Done and acceptance criteria
 4. Following the values of Scrum and enforcing the 3 pillars of Scrum in everyday work

3.3.2 Responsibilities and duties of a Product Manager

1. Driving the Product vision
 - a. By understanding the competitive landscape and appropriate implementation
 - b. By understanding the market conditions and change factors
 - c. By understanding the needs of the customer
 - d. By acknowledging input from the Product Owner
2. Building the high-level Product roadmap

- a. Collaborating with the customer to understand the current relevant initiatives and associated business objectives
 - b. Collaborating with the Product Owner to redefine the roadmap with respect to the work results of Developers
 - c. Reflecting the priority of high-level Product Backlog
 - d. Acknowledging restrictions and dependencies resulting from various aspects of the projected product timeline
 - e. Evaluating impact of ServiceNow support issues on the Product roadmap and updating it when necessary
3. Maximizing the value of the Product by supporting the Product Owner with the necessary input to build a qualitative potentially releasable product increment every sprint
 4. Defining high-level percentile allocation of the work to particular areas of the Product Backlog
 5. Maintaining relationships with the customers; building and cultivating trust with all stakeholders

3.3.3 Collaboration between the Product Owner and Product Manager

- If the Product Owner is not able to address an issue raised by Developers, he/she should consult the Product Manager and/or the customer with prior approval
- The Product Owner should support the Product Manager in building and maintaining the Product roadmap, by providing valuable input from the Developers and their work; the accountability remains with the Product Manager
- The Product Manager needs to closely collaborate with the Product Owner, for him to fully understand the Product vision and be able to explain it to the Developers
- The Product Owner supports the Product Manager in defining a proper percentile allocation of the work on various Product Backlog item categories needed to be done in the upcoming sprints, taking into account the work/product status and feedback from the Developers
- The Product Manager supports the Product Owner in resolving team or higher-level impediments if requested to do so, by using his/her knowledge, experience and capabilities
- Both collaborate on building a high-level Product Backlog with at most epic-level Product Backlog items (more in [5.3 - The Product Backlog \(see page 160\)](#)) and establishing a high-level prioritization
- Both should agree on a decision about releasing a ready product version, with accountability on the Product Manager
- The Product Owner should provide relevant input regarding ServiceNow issues from support to support the Product Manager's decision process during an assessment of the impact of the issue on the Product roadmap
- Both should collaborate to identify proper placement of ServiceNow issues from support in the Product Backlog structure by defining an existing or new epic as its parent

3.3.4 RACI matrix for EG Scrum events and sessions

	Product Owner	Product Manager
--	----------------------	------------------------

Backlog refinement	A	As the owner of the Product Backlog, the PO is accountable for any activities related to managing it. The PO must participate in the refinement together with the Developers to enable the team to conduct the event efficiently, resulting in better understanding of the Product Backlog by the team. The PO should provide Developers with clarifications and answers regarding any questions or to get back to the team as soon as possible, if he/she is unable to respond clearly.	C	The Product Manager can be consulted on all matters related to the act of refinement of the Product Backlog, including ServiceNow issues impact relevance on the Product roadmap
Sprint Planning	R	The Product Owner is co-responsible for conducting the Sprint Planning as part of the Scrum Team. The PO sets the goal(s) for the upcoming sprint and defines what stories could help in achieving that goal. The PO discusses with the Developers to clarify (when needed) Product Backlog items and negotiates the sprint backlog to meet the goal.	C	The Product Manager can be consulted on all matters related to the Sprint Planning event, in particular the priority of items in the Product Backlog.
Daily Scrum	C	The Product Owner can be consulted on all matters related to the team's Daily Scrum. The PO can be asked to join the event if needed and if it does not disturb the Developers.	I	The Product Manager should only be informed about the occurrences of the Daily Scrum.
Sprint Review	R	The Product Owner is responsible for the Sprint Review event. He/she is the moderator and drives the event. The PO may ask for support from the Scrum Master in conducting the event efficiently and in accordance with the best practices. The PO ensures Stakeholders are invited to the event.	C	The Product Manager can be consulted on all matters related to the Sprint Review event, in particular when the PO requests support regarding the presentation of the vision and review of the high-level product roadmap.
Sprint Retrospective	R	The Product Owner is responsible for participating in the Sprint Retrospective. As part of the Scrum Team, he/she should inspect the completed Sprint, identify action items and create an improvement plan for the next sprint together with the team. He/she may ask the Scrum Master for support in facilitating the event.	I	The Product Manager should only be informed about the occurrences of the Sprint Retrospective event, also especially about any outcome concerning the product ownership.

- ⓘ Responsible | Does the work to complete the task - at least 1 team member per task
- ⓘ Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
- ⓘ Consulted | Provides input to the work based on own expertise - no limit of team members
- ⓘ Informed | Needs to be kept in the loop - no limit of team members

3.3.5 Kanban

The role of the Product Owner in Kanban is similar to that in Scrum. Process framework and events are the only differences here. The main responsibility of Product Owner is still understanding the needs and expectations of customers, reflecting them by properly ordering work items in the backlog, and actively participating with the team in the process of clarifying the backlog items.

3.3.6 RACI matrix for optional Kanban meetings and sessions

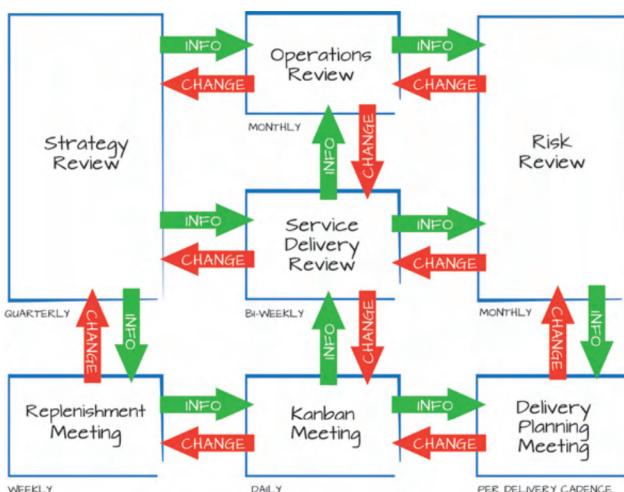
All Meetings in Kanban are optional and should be decided by the team which should be used. However, they might be valuable from the "Implementing Feedback loops" perspective. Kanban defines seven specific feedback opportunities or cadences. Cadences are the cyclical meetings and reviews that drive evolutionary change and effective service delivery. "Cadence" may also refer to the time period between reviews—one workday or one month, for example: Choosing the right cadence is context-dependent and it is crucial to good outcomes. Too-frequent reviews may compel changing things before seeing the effect of previous changes, but if they are not frequent enough, poor performance may persist longer than necessary.

Replenishment Meeting	A	This meeting is for moving items over the commitment point (and into the system) and to oversee the preparation of options for future selection.
The Kanban Meeting	C	This is the (usually) daily coordination, self-organization, and planning review for those collaborating to deliver the service. It often uses a "stand-up" format to encourage a short, energetic meeting with the focus on completing work items and unblocking issues.
Strategy Review	R	This is for the selection of the services to be provided and to define for this set of services the concept of "fit for purpose"; also for sensing how the external environment is changing in order to provide direction to the services.
Operations Review	R	This is to understand the balance between and across services, deploying resources to maximize the delivery of value-aligned with customers' expectations.
Risk Review	R	This review is to understand and respond to the risks to effective delivery of services; for example, through blocker clustering

Replenishment Meeting	A	This meeting is for moving items over the commitment point (and into the system) and to oversee the preparation of options for future selection.
Service Delivery Review	C	This is to examine and improve the effectiveness of a service (this and subsequent cadences apply to a single service).
Delivery Planning Meeting	R	This is to monitor and plan deliveries to customers.

- i** Responsible | Does the work to complete the task - at least 1 team member per task
 Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
 Consulted | Provides input to the work based on own expertise - no limit of team members
 Informed | Needs to be kept in the loop - no limit of team members

Implementing the seven cadences does not imply adding seven new meetings to an organization's overhead, although the Replenishment and Kanban Meetings are considered a baseline in nearly all Kanban implementations.



(A set of cadences showing feedback loops infographic - <https://kanban.university/>)

3.3.7 Small Scale Scrum

The Product Owner is an optional person in the Small Scale Scrum.

If the team has the capacity or within the team is a business person so such person can take responsibilities of Product Owner. Still, one thing needs to be highlighted. Small Scale Scrum emphasis for free communication on each line with the team and also between the team and developers.

3.4 3.4 - The role of a Scrum Master

NGA 2.0

In NGA 2.0 can be selected the best option for the team:

- EG Scrum
- Kanban
- Small Scale Scrum

And also we have gathered here best practices for teams who are struggling with:

- Consultancy Teams
- High Granularity Teams

The above approaches will help Agile Coaches and Business Units to fill out their needs in the Agile way of working.

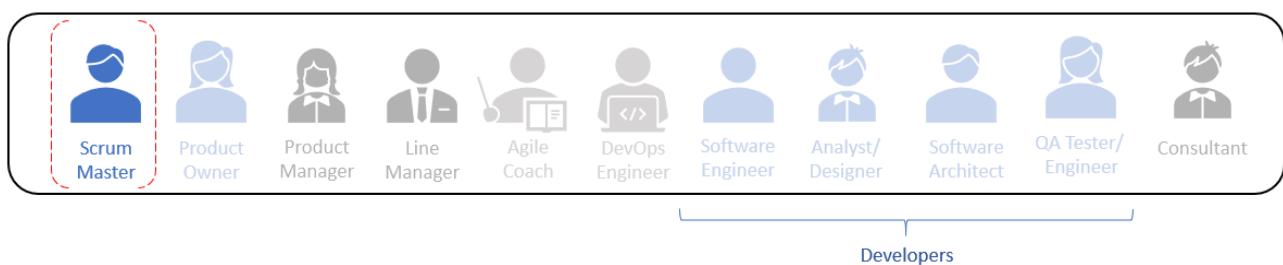
3.4.1 EG Scrum

The Scrum Master is accountable for establishing Scrum as defined in the Scrum Guide. They do this by helping everyone understand Scrum theory and practice, both within the Scrum Team and the organization.

The Scrum Master is accountable for the Scrum Team's effectiveness. They do this by enabling the Scrum Team to improve its practices, within the Scrum framework.

Scrum Masters are true leaders who serve the Scrum Team and the larger organization.

The Scrum Guide, SCRUM.org



3.4.2 Responsibilities and duties of a Scrum Master

1. Facilitating Scrum events
 - a. Showing understanding and practical knowledge of Scrum events and their purpose
 - b. Demonstrating proper, effective execution of Scrum events and related sessions
 - c. Finding the right balance between moderating the events and allowing teams to take ownership based on the team's level of maturity

- d. Sharing remarks and proposing best practices, improvements to the team's self-management during Scrum events and related sessions
 - e. Ensuring that all of the events take place together with the team in the form agreed by EG (more in [4 - Events and working together in sprints \(see page 107\)](#))
 - f. Managing Sprints (starting, closing) in Jira (more in [7.1.2 - Planning and starting a Sprint \(see page 254\)](#))
2. Driving team level improvements
- a. Leveraging team metrics, charts and dashboards to drive delivery progress and provide transparency through Jira and Confluence
 - b. Cultivating a culture of continuous improvement within the Scrum Team
 - c. Identifying areas for improvement within and surrounding the Scrum Team, including skills, tools and processes
 - d. Focusing on the needs of Developers and Product Owner, as a servant leader
3. Managing challenges and impediments
- a. Resolving team level impediments, which block the work of Developers on a daily basis
 - b. Escalating impediments which need to be addressed at a higher level to Agile Coaches or Managers
 - c. Identifying challenges, dependencies and risks related to skills, processes and tooling which may impact the team's work in the future; managing and helping the team address them in a transparent way
 - d. Shielding the team from external influence and interference which negatively impacts their performance
4. Coaching on Scrum values & supporting Scrum implementation and adaptation
- a. Training members of the Scrum Team to follow Scrum rules, enforce the 3 pillars of Scrum and live by the values of Scrum in everyday work
 - b. Coaching the surrounding organization on understanding empirical process control and awareness of Scrum rules and values
 - c. Coaching the Scrum Team in becoming a self-managed team capable of running Sprint events independently
 - d. Enforcing proper use of EG company-wide tools supporting agile software delivery, such as Jira, Confluence or BitBucket
5. Helping in understanding the product backlog, goals and supporting effective product backlog management
- a. Ensuring that the Developers understand the product backlog items
 - b. Ensuring that the product vision, the release and sprint goals are clear to both the Developers and the Product Owner alike
 - c. Helping the Product Owner and Product Manager find the best ways to manage and prioritize the Product Backlog, in order to achieve a maximal increase in resulting product value

3.4.3 RACI matrix for EG Scrum events and sessions

Backlog refinement	A/C	The Scrum Master can support the team by identifying optimal ways of conducting backlog refinement if requested by the team. As part of his/her responsibility to ensure proper understanding of the product backlog by the team, the Scrum Master can be accountable for enforcing the conduct of backlog refinement. Otherwise, he/she can be consulted on all matters related to the act of refinement.
Sprint Planning	A	The Scrum Master is accountable for the Sprint Planning to take place and for keeping it within the time-box. He/she trains the team on how to conduct the event and monitors progress while introducing improvements throughout the learning process. He/she ensures that no one impacts the sprint backlog scope except for the Developers. The Scrum Master should be aware at the end of the planning, how the team intends to meet the sprint goal.
Daily Scrum	A	The Scrum Master is accountable for the Daily Scrum to take place at the same time and location regularly; he/she keeps the event within the 15-minute time-box. He/she supports the team in training to conduct the event, for example using 3 default questions (more in 4.4 - The Daily Scrum (see page 120)). The Scrum Master ensures that no one disrupts the meeting on a daily basis.
Sprint Review	A	The Scrum Master is accountable for the Sprint Review to take place and for keeping it within the time-box. He/she trains the team on how to conduct the event and monitors progress while introducing improvements throughout the learning process. He/she enforces transparency throughout the Sprint Review and instructs participants about their responsibilities in scope.
Sprint Retrospective	A	The Scrum Master is accountable for the Sprint Retrospective to take place and for keeping it within the time-box. He/she ensures that the meeting is positive and productive by training on the proper implementation of the event, with practical examples. He/she can support in moderation especially during the phase of low team maturity. The Scrum Master participates in the Sprint Retrospective and encourages the whole team to seek & follow improvements.

- i** Responsible | Does the work to complete the task - at least 1 team member per task
 Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
 Consulted | Provides input to the work based on own expertise - no limit of team members
 Informed | Needs to be kept in the loop - no limit of team members

3.4.4 Kanban

For Kanban teams, Scrum Master (aka Flow Master) is accountable for establishing Kanban practices in the team:

- Visualization of the Workflow
- Limiting Work in Progress (WIP)
- Active management of work items in progress (Manage Flow)
- Make Policies Explicit (a process and flow policies like WIP limits, capacity allocation, Definition of Done)
- Implement Feedback loops (7 feedback opportunities/ optional Kanban meetings)
- Improve Collaboratively, Evolve Experimentally. Inspecting and adapting the team's Definition of Workflow

Additionally Scrum Master is responsible for the flow of work in delivering selected items to customers and for facilitating (optional) meetings.

3.4.5 RACI matrix for optional Kanban meetings and sessions

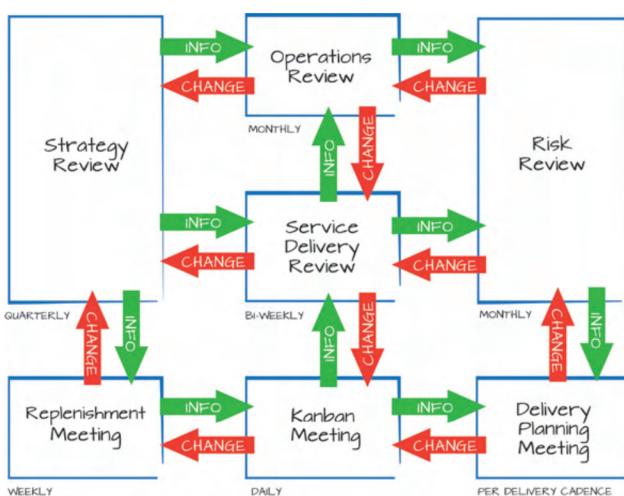
All Meetings in Kanban are optional and should be decided by the team which should be used. However, they might be valuable from the "Implementing Feedback loops" perspective. Kanban defines seven specific feedback opportunities or cadences. Cadences are the cyclical meetings and reviews that drive evolutionary change and effective service delivery. "Cadence" may also refer to the time period between reviews—one workday or one month, for example: Choosing the right cadence is context-dependent and it is crucial to good outcomes. Too-frequent reviews may compel changing things before seeing the effect of previous changes, but if they are not frequent enough, poor performance may persist longer than necessary.

Replenishment Meeting	A/C	This meeting is for moving items over the commitment point (and into the system) and to oversee the preparation of options for future selection.
The Kanban Meeting	A	This is the (usually) daily coordination, self-organization, and planning review for those collaborating to deliver the service. It often uses a "stand-up" format to encourage a short, energetic meeting with the focus on completing work items and unblocking issues.
Strategy Review	A	This is for the selection of the services to be provided and to define for this set of services the concept of "fit for purpose"; also for sensing how the external environment is changing in order to provide direction to the services.
Operations Review	A	This is to understand the balance between and across services, deploying resources to maximize the delivery of value-aligned with customers' expectations.
Risk Review	A	This review is to understand and respond to the risks to effective delivery of services; for example, through blocker clustering

Replenishment Meeting	A/C	This meeting is for moving items over the commitment point (and into the system) and to oversee the preparation of options for future selection.
Service Delivery Review	A	This is to examine and improve the effectiveness of a service (this and subsequent cadences apply to a single service).
Delivery Planning Meeting	A	This is to monitor and plan deliveries to customers.

- ⓘ Responsible | Does the work to complete the task - at least 1 team member per task
- ⓘ Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
- ⓘ Consulted | Provides input to the work based on own expertise - no limit of team members
- ⓘ Informed | Needs to be kept in the loop - no limit of team members

Implementing the seven cadences does not imply adding seven new meetings to an organization's overhead, although the Replenishment and Kanban Meetings are considered a baseline in nearly all Kanban implementations.



(A set of cadences showing feedback loops infographic - <https://kanban.university/>)

3.4.6 Small Scale Scrum

In Small Scale Scrum role of Scrum Master is optional. This approach puts emphasis on wide communication which will help to produce value.

Teams responsibilities that relate to Scrum Mastering:

- Conducting events
- Facilitation of events
- Impediments removal
- Wide communication with stakeholders

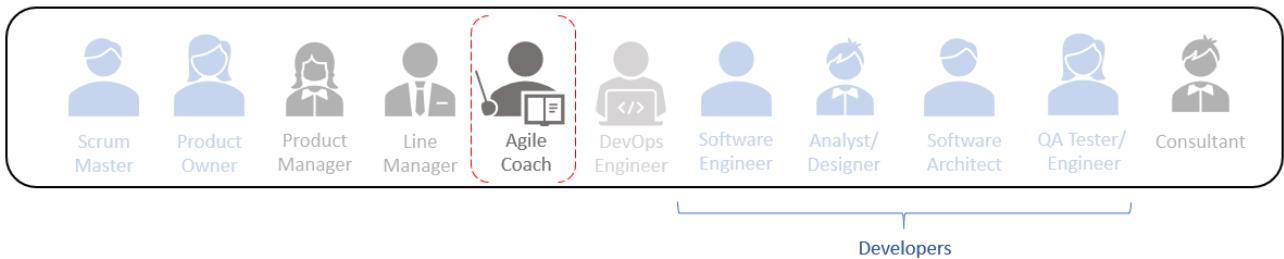
But if there is a space for the Scrum Master role one of the developers can partially act in this role.

3.5 3.5 - The role of an Agile Coach

The Agile Coach is responsible for fostering the agile mindset, spreading agile values and principles throughout the organization's working environment with a specific focus on continuous improvement.

The Agile Coach works to support building teams using Scrum as one of the frameworks, as well as to assess and drive the teams' maturities through mentoring and supporting their self-management. Agile Coaches “train the trainer” by working with the Scrum Masters to help them grow and become even better coaches and servant leaders for their teams.

Dariusz Szlek, EG Agile Coach



3.5.1 Roles and duties of an Agile Coach

1. Focusing on driving team-level agile practices, communication and efficiency
 - a. Training agile teams on proper adoption of EG Scrum and agile software development practices
 - b. Coaching agile teams on building team autonomy, improving self-management and product awareness
 - c. Improving team communication, relationships and collaboration through team-building workshops and creating a team identity
 - d. Teaching teams to enforce empiricism through building transparency around their work and incorporating inspection and adaptation into everyday product development
2. Contributing to the development and adoption of agile governance processes and tools across EG
 - a. Supporting DevOps with insight on tools as a connection point between the agile processes and the tooling which supports it
 - b. Evolving and adopting agile software development processes in EG
 - c. Creating a single point of truth for EG governance and rules
3. Evaluating and developing EG agile team maturity with respect to EG Scrum, agile values, principles and best practices

- a. Building and supporting new teams in becoming stable, performing EG Scrum teams
 - b. Building and improving the EG standard maturity model
 - c. Measuring EG teams' maturity against the model and focusing on improvement areas on a team-level
 - d. Optimizing team processes and governance around the EG Scrum framework with respect to agile values and principles
4. Coaching and mentoring Scrum Masters, Product Owners and Managers
- a. Working with Scrum Masters through "Train the trainer" in order to become even better servant leaders and teachers for their teams
 - b. Working with Product Owners/Managers to help understand agile product development and improve efficient backlog management techniques / best practices
 - c. Working with Managers to help understand their role in EG Scrum and how they can support Scrum Teams in becoming more efficient, developing as professionals, as well as delivering higher value to the customer
5. Spreading and cultivating Scrum values and the agile mindset throughout the EG organization
- a. Leading by example - teaching by doing - as a common practice in developing a mature agile mindset across the organization
 - b. Propagating the 5 values of Scrum, as well as agile values and principles in everyday work; explaining and elaborating the idea behind unclear statements / practices
 - c. Sharing knowledge and past experience to help improve the way EG works as a software development company

3.5.2 RACI matrix for EG Scrum events and sessions

Backlog refinement	C	The Agile Coach can be consulted for coaching and mentoring on proper execution, facilitation and best practices of running the meeting. He/she can provide hands-on support, examples especially at the team's starting point or when the team represents low maturity in this particular area or as long as the Scrum Master requires assistance.
Sprint Planning	C	The Agile Coach can be consulted for coaching and mentoring on proper execution, facilitation and best practices of running the event. He/she can provide hands-on support, examples especially at the team's starting point or when the team represents low maturity in this particular area or as long as the Scrum Master requires assistance.
Daily Scrum	C	The Agile Coach can be consulted for coaching and mentoring on proper execution, facilitation and best practices of running the event. He/she can provide hands-on support, examples especially at the team's starting point or when the team represents low maturity in this particular area or as long as the Scrum Master requires assistance.

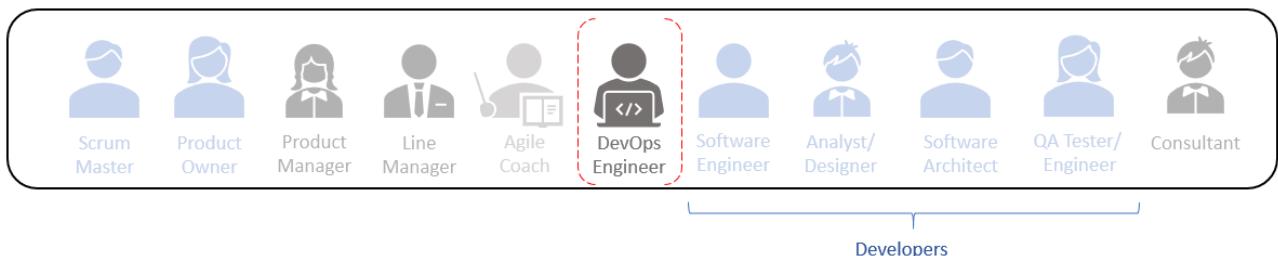
Sprint Review	C	The Agile Coach can be consulted for coaching and mentoring on proper execution, facilitation and best practices of running the event. He/she can provide hands-on support, examples especially at the team's starting point or when the team represents low maturity in this particular area or as long as the Scrum Master requires assistance.
Sprint Retrospective	C	The Agile Coach can be consulted for coaching and mentoring on proper execution, facilitation and best practices of running the event. He/she can provide hands-on support, examples especially at the team's starting point or when the team represents low maturity in this particular area or as long as the Scrum Master requires assistance.

- (i) Responsible | Does the work to complete the task - at least 1 team member per task
 Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
 Consulted | Provides input to the work based on own expertise - no limit of team members
 Informed | Needs to be kept in the loop - no limit of team members

3.6 3.6 - The role of a DevOps Engineer

DevOps is a practice of combining software development (Dev) and information-technology operations (Ops). It's a way of working, which aims to shorten the systems development life cycle and provide continuous delivery, increasing the software quality at the same time.

Jarosław Krochmalski, DevOps Lead

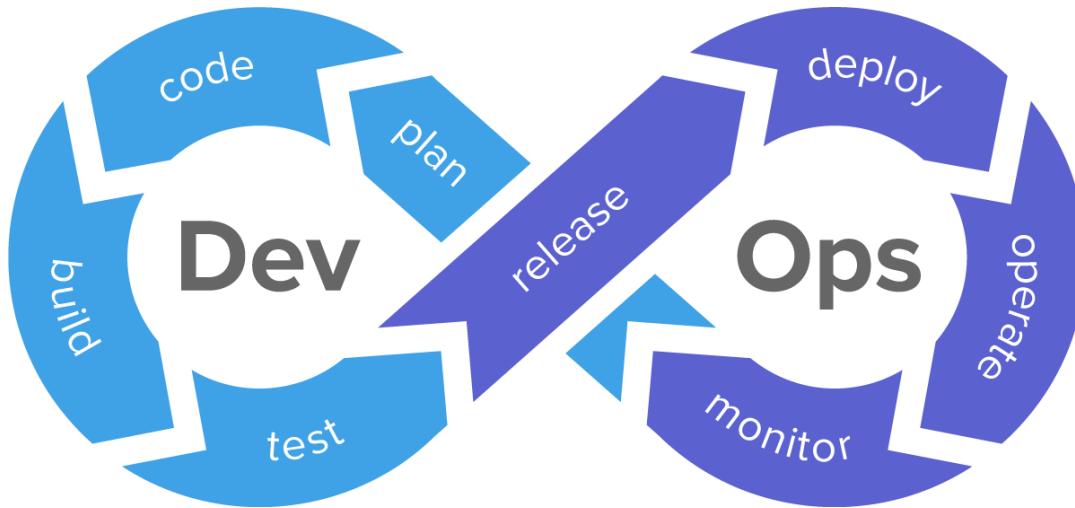


DevOps culture enables the Organization to achieve Agility and to meet the dynamic expectations of customers. As DevOps is intended to be a cross-functional mode of working, those that practice the methodology use different sets of tools - referred to as toolchains, rather than a single one. These toolchains are expected to fit into one or more of the following categories, reflective of key aspects of the development and delivery process:

1. **Coding** – code development and review, source code management tools, code merging
2. **Building** – continuous integration. Triggering build process when there is a code change.
3. **Testing** – continuous testing tools that provide quick and timely feedback on potential issues

4. **Packaging** – artefact repository, application pre-deployment staging
5. **Releasing** – change management, release approvals, release automation
6. **Configuring** – infrastructure configuration and management, infrastructure as code tools
7. **Monitoring** – application performance, end-user experience

Agile and DevOps serve complementary roles: several standard DevOps practices such as automated build and test, continuous integration and continuous delivery originated in the Agile world, which dates (informally) to the 1990s, and formally to 2001. Agile can be viewed as addressing communication gaps between customers and Developers, while DevOps addresses gaps between Software Engineers and IT operations/infrastructure. Also, DevOps is focused on the deployment of developed software, whether it is developed via Agile or other methodologies.



3.6.1 Responsibilities and duties of a DevOps Engineer in EG

A DevOps Engineer can be either a team member or a delegated engineer to work closely with Developers. The main duties include:

1. Supporting the Team in day to day operations, that includes working with:
 - a. issue management system
 - b. version control server
 - c. automation/build server
 - d. artefact repository
 - e. release management
 - f. environment configuration and management
 - g. monitoring applications
2. Understanding the needs and challenges the Team has with operations and development, and partner with the Team to formulate solutions that support their goals.
3. Helping the Team to develop solutions and processes for Continuous Integration / Continuous Delivery

4. Maintaining the strong expertise and knowledge of current and emerging processes, techniques and tools.
5. Being a part of DevOps Community of Practice, which encourages communication, knowledge-sharing and continuous learning.

3.6.2 RACI matrix for EG Scrum events and sessions

Backlog refinement	C	A DevOps Engineer can be consulted for support on CI/CD, automation and tooling with respect to the Product Backlog items being evaluated. He/she can provide hands-on examples and ideas relevant to the implementation and maintenance of the Product Backlog items being refined. A DevOps Engineer can collaborate with the team to ensure solution quality around the CI/CD and automation frameworks.
Sprint Planning	I	A DevOps Engineer may be informed about the Sprint Planning event but does not participate in any way unless specified otherwise, case by case.
Daily Scrum	C	A DevOps Engineer can be consulted for support on CI/CD, automation and tooling with respect to the Sprint Backlog of the team. He/she can provide hands-on examples and ideas relevant to the implementation and maintenance of the Sprint Backlog items being delivered in the ongoing sprint. A DevOps Engineer can collaborate with the team to ensure solution quality around the CI/CD and automation frameworks.
Sprint Review	I	A DevOps Engineer may be informed about the Sprint Review event but does not participate in any way unless specified otherwise, case by case.
Sprint Retrospective	I	A DevOps Engineer may be informed about the Sprint Retrospective event but does not participate in any way.



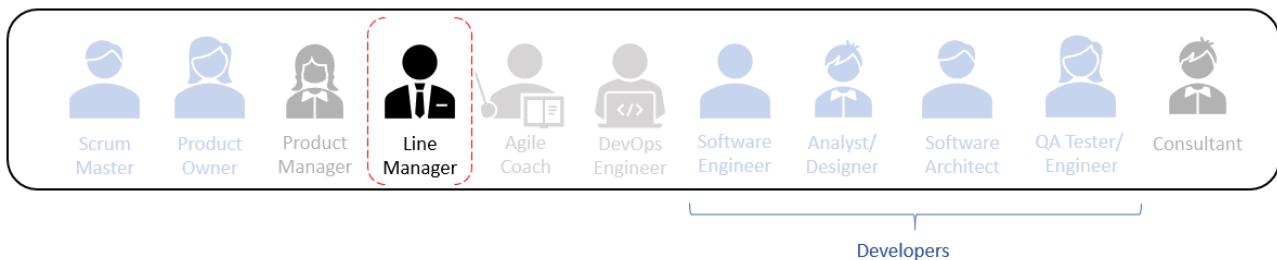
- Responsible | Does the work to complete the task - at least 1 team member per task
- Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
- Consulted | Provides input to the work based on own expertise - no limit of team members
- Informed | Needs to be kept in the loop - no limit of team members

3.7 3.7 - The role of a Manager

The role of Management in EG Scrum differs significantly from a traditional management approach. The focus of managing should shift from work items and their fulfillment to the teams/persons doing the work and the results, which they achieve.

Managers need to support their teams in clarifying their goals as a team and as individuals, while providing the means, tools and opportunities to reach those goals. The role of a Manager should embrace Scrum's servant-leadership, pillars and values in order to allow teams to thrive, by following a coherent strategic vision of the organization.

Dariusz Szlęk, EG Agile Coach



3.7.1 Responsibilities and duties of a Manager

1. Drawing a long-term vision/mission for Scrum Teams
 - a. Collaborating with Department Heads, Product Owners and Product Managers on aligning the vision/mission statements
 - b. Setting milestones for the Scrum Teams in achieving higher maturity
2. Promoting and enforcing agility and its importance within the organization
 - a. Working with other managers and stakeholders to strengthen the awareness of EG Scrum and the Agile Manifesto across the organization
3. Building and protecting the personal, psychological safety of the managed subordinates
 - a. Organize workshops about the Non Violent Communication and Open culture feedback
 - b. Lead by an example in following Agile / Scrum values and sharing those in organization
4. Managing operational and logistical needs of Scrum Teams
 - a. Providing necessary resources for the teams
 - b. Providing or supporting the acquisition of necessary means of travel and accommodation
 - c. Ensuring proper work environment space
 - d. Resolving all employment requests, including leaves and absences
5. Fostering and enabling personal/professional skills development within the Scrum Teams
 - a. Proposing paths for personal growth and self-development

- b. Providing means for developing soft skills and technological excellence, including internal/external training and/or workshops
 - c. Evaluating progress, providing constructive feedback and guiding towards set goals
6. Providing support to the team in order to tackle identified issues on their own if asked

3.7.2 RACI of Agile Leads responsibilities

Activities	Responsible	Accountable	Consulted	Informed
Hiring process (organization)	HR	Manager	PO, SM	PM
Hiring process (check)	Team	Manager	PO, SM	PM
Team vision / mission	Manager	Manager	PO, PM	SM, Team
Milestones for a team	Manager	Manager	PO, PM	SM, Team
Team growth develop	Team	Manager	PO, PM	SM
Work environment (setup)	Manager	Manager	Team, SM	PO, PM
Work environment (usage)	Team	Manager	SM, PO	PM
Sick leave registration	Team	Manager	SM	PO
Agile change agents	Manager, SM, AC	Agile Coach	SM, AC	All
Spread open culture	Manager, SM, AC	Agile Coach	SM, AC	All

3.7.3 RACI matrix for managers in EG Scrum events

Backlog refinement	I	A Manager may be informed about the Backlog refinement sessions but does not participate in any way unless specified otherwise, case by case.
---------------------------	---	---

Sprint Planning	I	A Manager may be informed about the Sprint Planning event but does not participate in any way unless specified otherwise, case by case.
Daily Scrum	I	A Manager may be informed about the occurrences of the Daily Scrum event but does not participate in any way unless specified otherwise, case by case.
Sprint Review	I	A Manager may be informed about the Sprint Review event but does not participate in any way unless specified otherwise, case by case. Occasionally, the Manager may be an observer at the Sprint Review. An exception occurs when a Manager fulfills a stakeholder role, in which case he/she acts as a customer represented by the Product Owner.
Sprint Retrospective	I	A Manager may be informed about the Sprint Retrospective event but does not participate in any way. A Manager may be asked to support the team in resolving issues resulting from the conducted Sprint Retrospective.

3.7.4 Splitting leads responsibilities in Scrum

Activities	Responsible	Accountable	Consulted	Informed
Strategy / Vision	Product Manager (PM)	Product Manager (PM)	PO, BO's	Team
New initiatives	PO, PM, BO's	Business Owners (BO)	PO, PM, BO's	Team
Backlog preparation	Product Owner / team	Product Owner (PO)	PM, SM	BO's
Planning (PI, sprints)	Team / Product Owner	Team	PM, SM	BO's
Delivery progress	Team	Team	PO, SM	PM, BO's
Remove impediments	Scrum Master / team	Scrum Master (SM)	AC, manager	PO, PM
Agile workshops	Scrum Master / AC	Agile Coach (AC)	SM / AC	PO, PM

Technical excellence	Team (cross-functional)	Team (cross-functional)	Architect, PO	PM
Plan / manage work	Team (self-organizing)	SM (supporting team)	Agile Coach	PO, PM
Deliver improvements	Team (self-organizing)	SM (supporting team)	Agile Coach	PO, PM
Personal conflicts	Team members, SM	SM (NVC, mediator)	AC, manager	-
Product innovation	Team self-managing	PO, PM	SM	-

- i** Responsible | Does the work to complete the task - at least 1 team member per task
 Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
 Consulted | Provides input to the work based on own expertise - no limit of team members
 Informed | Needs to be kept in the loop - no limit of team members

3.7.5 Anti-patterns with manager role in Scrum

Backlog refinement	A Manager may insist to work on not refined items as they are important for business stakeholders which lowers efficiency. Setting up priorities are up to PO.
Planning (PI, sprint)	A Manager tries to "motivate" teams to plan more than average Velocity and without buffer which lower productivity and believing in self-managing.
Daily Scrum	A Manager participates actively in daily proposing actions. Presence of a Manager might limit openness on daily and change it to status update meeting.
Sprint Review	A Manager starts to ask questions during review why team did not achieve goals which demotivates team to present achievements and should be part of retro.
Sprint Retrospective	A Manager is attending team retrospective which limits openness from team members. Retro is dominating by a Manager who tells SM & team what to do.

Manager as a team member	A Manager is at the same time team member, which can make atmosphere not so open and not promotes self-managing according to previous managing style.
Manager as a Scrum Master	A Manager who turned to be Scrum Master can find new role very challenging as SM is an opposite role. SM can't manage people but empower self-managing.
Manager as a Product Owner	A Product Owner with managerial role (or approach) can limit challenging lack of value in proposed priorities or ideas for different implementation to achieve goal.

3.8 3.8 - The role of a Consultant

NGA 2.0

In NGA 2.0 can be selected the best option for the team:

- EG Scrum
- Kanban
- Small Scale Scrum

And also we have gathered here best practices for teams who are struggling with:

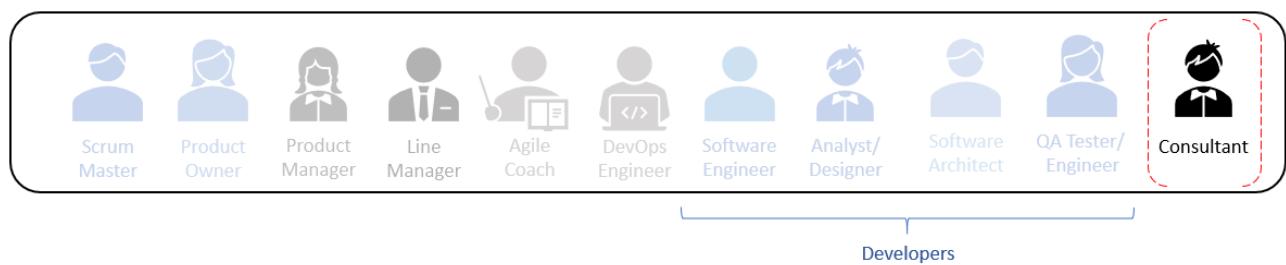
- Consultancy Teams
- High Granularity Teams

The above approaches will help Agile Coaches and Business Units to fill out their needs in the Agile way of working.

Apart from the Scrum Team in EG we are having the Consultant role which is usually the bridge linking the customers with the Scrum Team. This role is not part of the Scrum setup, but is very important in the areas of requirements creation and clarification, bugs raising and clarification. The Consultants may also support the Product Owner in the backlog prioritization and ordering.

Across EG we may have a different sets of Consultants role responsibilities. We may also have a situations in which Consultants are part of the Scrum Team taking i.e. the shared role of Developer and Consultant.

Cezary Perendy, EG Agile Coach



3.8.1 Responsibilities and duties of a Consultant in EG

The duties and responsibilities of the Consultants may differ across Business Units and teams in EG, but usually, the role covers the following portfolio:

1. Customer care - usually the first line of contact with customers
2. Handling customers requests:
 - bugs rising
 - reproducing the bugs in our environment
 - analysis of the customers requests, wishes, and expectations
 - clarification of customer expectations
 - participating in the estimation and approval of the estimation process
3. sales and offering process participation and support
4. systems setup and configuration for the specific customers
5. collaboration with the Product Owner and Developers in the areas of the refinement process and backlog prioritization
6. Alignment with the customers about the current status and progress of work

The Consultants, if they are participating in the development process or Product increment creation, may have a shared role (part-time) as Developers (with all the duties, responsibilities, and Scrum process participation aspects of such role described in the Playbook). In such a situation, it is recommended to specify (at least per Sprint) the ratio of the engagement/participation in the Scrum Team vs. Consultancy activities, to be able to properly divide the attention and align the available capacity.

3.8.2 RACI matrix for EG Scrum events and sessions

Backlog refinement	I/C	A Consultant should be informed about the Backlog refinement sessions and the scope of the issues being refined. If needed or valuable for the Backlog Refinement process, the Consultants may participate in it to provide the input, clarify the requirements and gather the needed information from customers.
Sprint Planning	I	A Consultant may be informed about the Sprint Planning event but does not participate in any way unless specified otherwise, case by case.
Daily Scrum	I	A Manager may be informed about the occurrences of the Daily Scrum event but does not participate in any way unless specified otherwise, case by case.
Sprint Review	I/C	A Consultant may be informed about the Sprint Review event. It is up to the Consultant to decide if he/she would like to participate in the event as a stakeholder.

Backlog refinement	I/C	A Consultant should be informed about the Backlog refinement sessions and the scope of the issues being refined. If needed or valuable for the Backlog Refinement process, the Consultants may participate in it to provide the input, clarify the requirements and gather the needed information from customers.
Sprint Retrospective	I/C	A Consultant may be informed about the Sprint Retrospective event but does not participate in any way. In specific situations, when the retrospective topics are referring to Consultants and Scrum Team collaboration, Consultant may be invited to the corresponding part of the Retrospective event to discuss the topic and participate in the creation of the improvements.

The above matrix refers only to the Consultant role. If the Consultant is sharing the role with one of the other roles described in the playbook, he/she should refer to the corresponding role as well to have the full overview of the RACI matrix.

- ⓘ Responsible | Does the work to complete the task - at least 1 team member per task
- ⓘ Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
- ⓘ Consulted | Provides input to the work based on own expertise - no limit of team members
- ⓘ Informed | Needs to be kept in the loop - no limit of team members

3.9 3.9 - EG Scrum roles in various contexts

Scrum roles can be observed from different perspectives when switching context among other elements of Scrum, such as: artifacts, events and relationship to other roles. Their interpretation across mutual relationships can clarify the understanding behind them and expectations towards them when working with agile product development.

3.9.1 General EG Scrum roles RACI matrix with respect to Scrum events and activities

- ⓘ Responsible | Does the work to complete the task - at least 1 team member per task
- ⓘ Accountable | Delegates work and is the last one to review it - limit to 1 team member per task
- ⓘ Consulted | Provides input to the work based on own expertise - no limit of team members
- ⓘ Informed | Needs to be kept in the loop - no limit of team members

Role / Event or activity	Backlog refinement	Sprint Planning	Daily Scrum	Sprint Review	Sprint Retrospective
Scrum Master	A/C	A	A	A	A
Product Owner	A	R	C	R	R
Product Manager	C	C	I	C	I
Manager	I	I	I	I	I
Agile Coach	C	C	C	C	C
DevOps Engineer	C	I	C	I	I
Software Engineers	R	R	R	R	R
Business Analyst	R	R	R	R	R
UX Designer	R	R	R	R	R
Consultant	I/C	I	I	I/C	I/C
SW Architect	R	R	R	R	R

3.9.2 Collaboration and mutual responsibility matrix for EG Scrum roles

---	Scrum Master	Product Owner	Product Manager	Manager	Agile Coach	DevOps Engineer	Software Engineers	Business Analyst / UX Designer	Consultant
-->									

Scrum Master	X	Supports in understanding Scrum, especially Product Backlog Management	Supports in understanding Scrum strengthens collaboration with the customer, escalates Product level impediments	Supports in understanding Scrum, provides feedback to better serve the scrum Team	Provides feedback to better support the scrum team and organization	Facilitates collaboration and communication with the scrum team	Supports the scrum team in any way necessary by the team, including mentoring, coaching and facilitation	Supports the scrum team in any way necessary by the team, including mentoring, coaching and facilitation	Supports in understanding Scrum may request participation in the refinement process and other events if needed
Product Owner	Follows the values and rules of Scrum, informs about possible impediments	X	Supports in managing the Product roadmap and providing feedback to work allocation to categories	Provides feedback to better serve the Scrum Team	Provides feedback to better support the scrum team and organization	Provides feedback regarding CI/CD and automation – their impact on the Product / Product Backlog	Clarifies any questions or doubts regarding the Product Backlog, prioritizes the Product Backlog, prioritizes the Product Backlog, prioritizes the Product Backlog,	Clarifies any questions or doubts regarding the Product Backlog, prioritizes the Product Backlog, prioritizes the Product Backlog,	Can organize the dialog regarding issues clarification and the Refinement process, May request consultation in the Product Backlog ordering

Product Manager	Resolves impediments on Product level, engages the customer into close collaboration	Explains the Product vision, resolves impediments on Product level, contact with the customer	X	Provides feedback from product perspective, supports in resolving impediments	Provides feedback to better support the Product Owner, the organization and the collaboration with the customer	Provides feedback regarding CI/CD and automation – their impact on the Product roadmap / high-level items	Supports the Product Owner if necessary	Supports the Product Owner if necessary	Can consult the high-level plan and requirements.
Manager	Resolves escalated impediments	Supports the Scrum Team if necessary in all scope	Provides feedback from management perspective, supports in resolving impediments	X	Provides feedback to better support the Scrum Team and the organization	Mentors, gathers feedback and supports self-development	Mentors, gathers feedback and supports self-development	Mentors, gathers feedback and supports self-development	Resolves escalated impediments

Agile Coach	Train the trainer, coaches and mentors to better fulfill the Scrum Master role	Supports in Scrum and agile product development practices, helps understand the importance of customer involvement	Supports in Scrum and agile product development practices, helps to focus the support on right areas of the Scrum Team and organization	Supports in Scrum and agile product development practices, helps to focus the support on right areas of the Scrum Team and organization	X	Provides training, coaching and mentoring with respect to Scrum, agile software development processes and tooling; facilitates collaboration and communication with other roles	Provides training, coaching and mentoring with respect to Scrum, agile software development processes and tooling; facilitates collaboration and communication with other roles	Provides training, coaching and mentoring with respect to Scrum, agile software development processes and tooling; facilitates collaboration and communication with other roles	Supports in Scrum and agile product development practices helps understand and the importance of customer involvement
--------------------	--	--	---	---	---	---	---	---	---

Dev Ops Engineer	Provides feedback from DevOps perspective, reports impediments, acknowledges suggestions and recommendations regarding followin g EG Scrum rules	Supports Product Backlog management and refinement with regard to CI/CD and automation, delivers necessary elements of the Product Backlog items	Provides feedback through the Product Owner if necessary	Provides feedback to better serve the Scrum Team, follows recommendations for self-development	Provides feedback to better support the Scrum Team and the organization	X	Provides insight to the CI/CD/automation flow to clarify Product Backlog item dependencies or expectations, provides feedback regarding software development	Provides insight to the CI/CD/automation flow to help better reflect the impact on functionalities and design, helps understand the technological impact	X
Software Engineers / Software Architect	Provides feedback from Software Engineers' perspective, reports impediments acknowledges suggestions and recommendations regarding followin g EG Scrum rules	Delivers a potentially releasable product increment at the end of the sprint, works on better understanding Product Backlog items and provides feedback on the Product	Provides feedback through the Product Owner if necessary	Provides feedback to better serve the Scrum Team, follows recommendations for self-development	Provides feedback to better support the Scrum Team and the organization	Provides insight to the Product Backlog items to clarify CI/CD/automation dependencies or expectations, provides feedback regarding DevOps flow operation	X	Provides technical insight to the Product Backlog items to make technological restrictions and dependencies transparent with respect to the functional specification or design	Provides feedback through the Product Owner if necessary

Business Analyst / UX Designer	Provides feedback from Business/UX perspective, reports impediments, acknowledges suggestions and recommends regarding followin g EG Scrum rules	Contributes to delivering a potential ly releasable product increment at the end of sprint, works on better understanding and refining Product Backlog items and provides feedbac k on the Product	Provides feedba ck through the Product Owner if necessary	Provides feedba ck to better serve the Scrum Team, follows recom mendations for self- development	Provides feedba ck to better support the Scrum Team and the organization	Provides functional / design support to improve understand ing of the target solution concept to better adjust the DevOps flow	Provides functional / design support to improve understand ing of the target solution concept to better match the target impleme ntation with respect to the accepta nce criteria	X	may request support regardi ng the refinem ent process , issues clarifica tion and gaps fill in.
Consultant	Help understand the customer needs and relationships	Supports the PO in the issues clarificat ion and Refinement process, can provide the input regardin g Backlog ordering and prioritiza tion	Can consult the high- level plan and requirements.	Provides feedba ck and suggest ions regardi ng the work organiz ation.	Can provide the customer perspe ctive context , as well as the participation and interfer ences in toward s the Scrum proces s,	can consult the related field as a subject matter expert	Can support the clarificat ion of the issues as well as custome r feedbac k/input gatherin g	Can support the clarificat ion of the issues as well as custome r feedbac k/input gatherin g	X

3.9.3 Ownership and responsibility matrix for Scrum artifacts across EG Scrum roles

	Product Backlog	Sprint Backlog	Definition of Ready & Done	Velocity & Capacity
Scrum Master	supports transparency, supports to maintain a healthy structure	supports transparency and ensures understanding	creates and adheres to, unless defined by the organization - then follows and expands to make more precise with time	supports transparency, draws conclusions and makes suggestions
Product Owner	manages, prioritizes and owns	defines goals		acknowledges and respects
Software Engineers / Software Architect	refines and clarifies to understand	creates, owns and maintains		creates, owns and maintains
Business analyst / UX designer	refines and clarifies to understand	creates, owns and maintains		creates, owns and maintains
Product Manager	supports from high-level / roadmap perspective	acknowledges and respects	contributes to defining on organizational level	acknowledges and respects
Agile Coach	consults	consults	consults and contributes to defining on organizational level	consults
Consultant	consults	if needed - can review	not applicable	not applicable

3.9.4 Kanban

For Kanban teams, roles and events are optional. Collaboration and mutual responsibilities for established roles are similar to EG Scrum. Please refer to mentioned matrix accordingly.

4 4 - Events and working together in sprints

- [4.1 - What an EG cadence looks like \(see page 107\)](#)
- [4.2 - The Backlog Refinement \(see page 113\)](#)
- [4.3 - The Sprint Planning \(see page 117\)](#)
- [4.4 - The Daily Scrum \(see page 120\)](#)
- [4.5 - The Sprint Review \(see page 122\)](#)
- [4.6 - The Sprint Retrospective \(see page 126\)](#)
- [4.7 - Daily work in a Scrum Team \(see page 128\)](#)
- [4.8 - Team building \(see page 136\)](#)
- [4.9 - Events for Kanban teams \(see page 143\)](#)

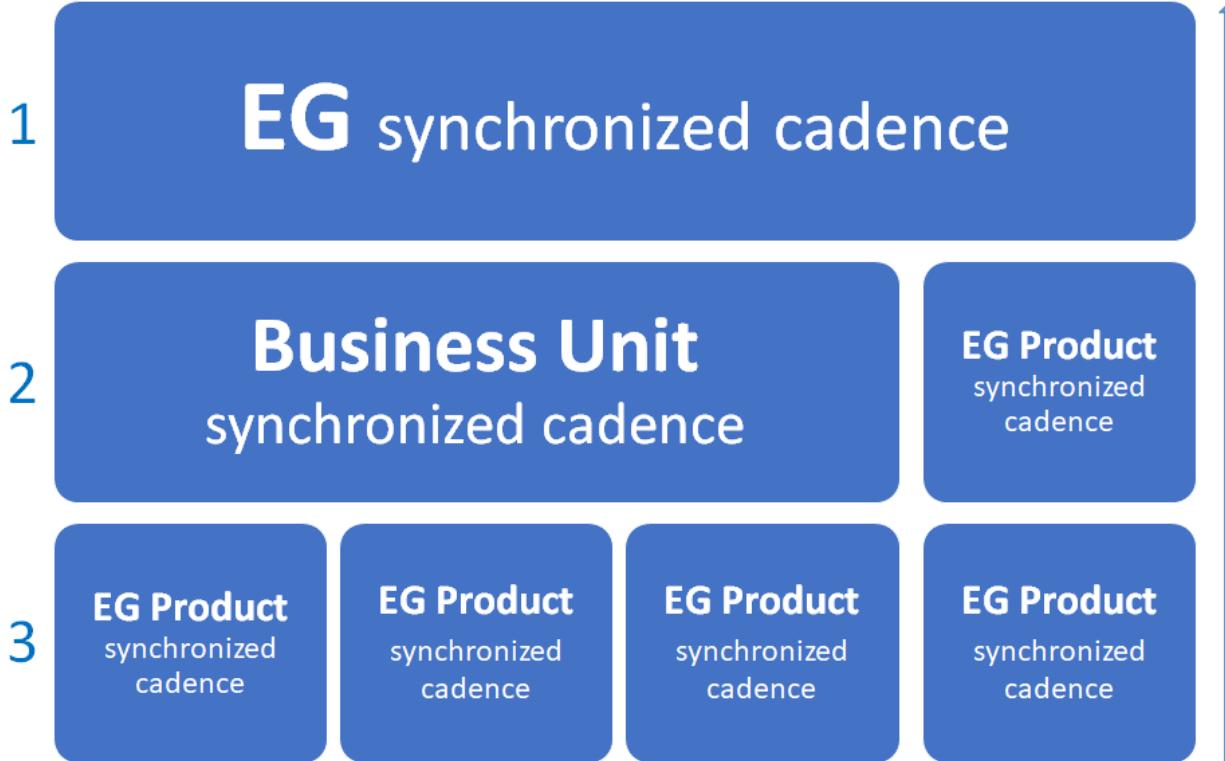
4.1 4.1 - What an EG cadence looks like

Independent of the selected agile product delivery approach, there should be a cadence of some sort defined for a team to conduct its routine operation. In EG Scrum and in Small Scale Scrum, that cadence will be defined by the Sprint cadence, while in KanBan there is a term called the KanBan cadence which is defined by the duration of the feedback loop, which implements the team/service-oriented events. What is relevant is that any cadence should be consistent and stable for the entire team. Below you will find a further overview of the cadence concept in EG.

4.1.1 Synchronized cadence in EG

The default cadence in EG is **2 weeks** long and is **synchronized** to foster awareness throughout the EG community, as well as simplify organization and eliminate mutual communication issues during Scrum events. All EG Sprints should start and be completed on **Tuesdays or Wednesdays** (of the same week across teams). In the case of KanBan service-oriented events, the cadence could be 1 week long as well if the feedback loop requires it; nevertheless, the timing of the cadence changed should be aligned analogically within the organizational unit.

The journey to a synchronized sprint cadence across EG is one passing through 3 levels of the organization structure and should be considered a learning path to reaching such a demanding yet beneficial setup.



- **No level** - lack of any synchronization of team cadences in an organization is highly not recommended, as it may lead to (among other problems and challenges):
 - highly unmanageable dependencies between teams
 - lack of a single, releasable Product Increment for a Product developed by more than 1 team
 - lack of awareness and calendar/timing conflicts between teams
 - issues with stakeholder availability

The lack of any synchronization shows a low agile maturity of the organization in terms of a single, functioning organism as a whole. While reaching global corporate synchronization is a big challenge and a long journey, small steps from the very beginning can help to build a path towards reaching higher levels of coherency.

- **Level 3** - cadence synchronization on a Product level is mandatory to achieve an effective team setup, conducting work on a single Product Backlog. This is the basis, with which all of the onboarded Teams should start from. Of course, not all teams will need to comply, if the team-product relation is 1 to 1. This has a highly significant influence on multiple-team products as well as single teams working on multiple products, where the cadence needs to be synchronized between the EG Products to fit the single team.
- **Level 2** - cadence synchronization on a Business Unit level is the next step to achieving EG-wide synchronized cadence. In some cases, this will be frequently possible from the very beginning for small and versatile BUs. In other cases reaching a synchronized cadence, will be a joint effort of independent Products within the Business Unit. In the beginning, working with synchronization on a Product level may be less strict and allow the teams to become used to the new tools and processes; after they become more mature this is definitely one of the milestones to reach on their road to proficiency.

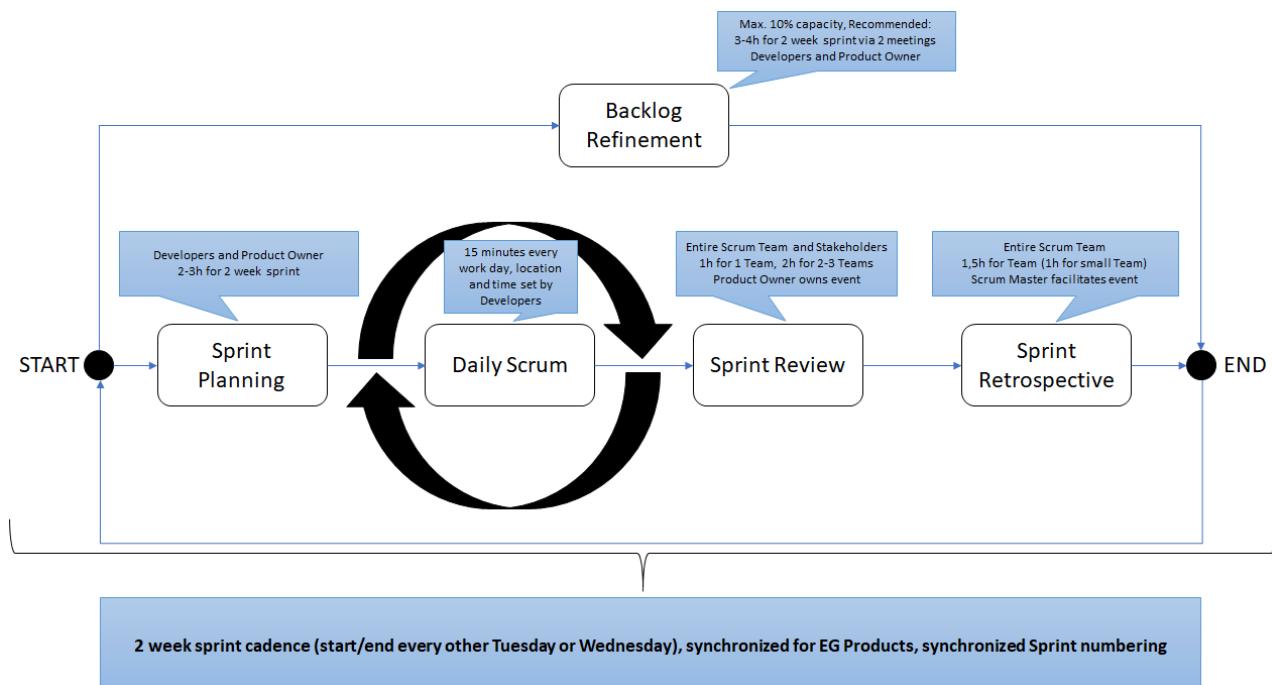
- **Level 1** - cadence synchronization on a corporate level in EG is the ultimate goal for the organization and may take some time to achieve. This goal gives way to the team-building and team maturity growth processes which will take place in the first sprints of the onboarded Teams. While synchronization brings multiple benefits on an organizational level, it brings the highest value only when the teams in the organization have already reached some level of stability and are able to continue improving on a not only internal but external (corporate) level as well.

4.1.1.1 The benefits of a synchronized cadence

To understand the value behind a synchronized sprint cadence, the following points can be considered:

- propagation of stable, consistent sprint change dates or KanBan cadence cycle throughout EG helps in
 - raising awareness about the Agile Events and Ceremonies in EG
 - developing a sense of importance around these events/ceremonies
 - spreading a regularity-based attitude among teams, managers and stakeholders
- a synchronized cadence fosters a respectful mindset around the event dates and the people, who participate - it is easier to plan any major, EG events or ceremonies around those dates, as they are known ahead of time and concern all (or most) teams
- with regular and synchronized sprint/cycle changes, it is easier to ensure the participation of crucial (and usually very occupied) stakeholders, into the Sprint Review or Service Delivery Review events
- synchronized cadences also allow to properly identify, address and mitigate cross-team (per Product) or even cross-product dependencies; this frequently leads to resource consolidation and optimization

4.1.2 EG Scrum



4.1.2.1 A common sprint naming convention

In order to simplify and raise coherency across EG, the Sprint numbering should be synchronized and unified. This should help to uniquely define *the product, the iteration and the team*.

For example:

A4 TMS Team 1 - S1_20 (19/11) or HOU Vision - S2_20 (04/02)

(i) EG Sprint naming convention: <Jira project code> <Team name> - S<#>_<2 year digits> (dd/mm)

4.1.2.1.1 Exceptions

Should any Scrum Team or Business Unit feel, that either the Sprint cadence duration or synchronized cycle negatively impacts their productivity, it should be consulted with an Agile Coach in order to:

- discuss the justification for altering the sprint duration by providing concrete and irrefutable reasoning
- discuss the possibility of shifting or expanding the Sprint change cycle due to a valid and justified reason (i.e. potential conflicts for some roles participating in multiple team events)

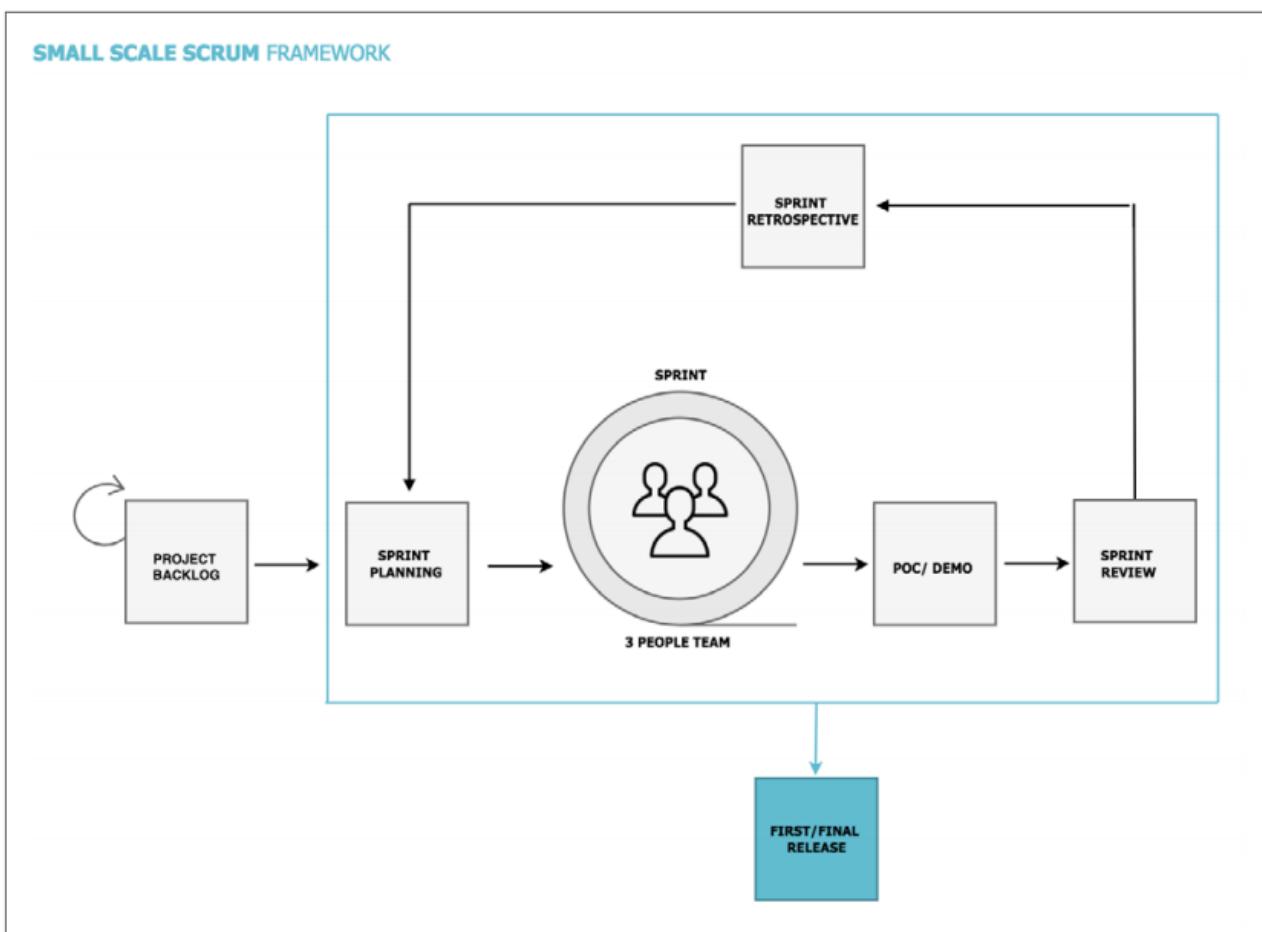
4.1.2.2 Sequence of events

The Sprint sequence of events in EG follows the clear definition of the Scrum Guide:

- a Sprint starts with the **Sprint Planning** ([4.3 - The Sprint Planning \(see page 117\)](#)) event, to forecast the work to be done by the Developers as agreed with the Product Owner; the event should be **2-3 hours** long and the following roles should participate:
 - Product Owner
 - Developers (including Software Engineers, Business Analysts and UX designers)
 - Scrum Master for the facilitation of the event
 - (optional) any additional experts and/or specialists required by the team to support a better understanding of the Product Backlog and who can optimize the value of the Product
- **Daily Scrum** ([4.4 - The Daily Scrum \(see page 120\)](#)) is executed every day by the Developers; the event should last **no more than 15 minutes**. Although the time and location of the event should be specified by the Developers, it is recommended to conduct the event at the beginning of the day at a constant location (for co-located teams). The Developers may ask others to join the event however, they should not be distracted during the timebox.
- a Sprint is summarized by the **Sprint Review** ([4.5 - The Sprint Review \(see page 122\)](#)) event, to elaborate on what was and wasn't done during the sprint and to present a working Demo to the Stakeholders. Everyone participates in the Sprint Review. The event should be timeboxed to **1 or 2 hours**, depending on how many teams participate in reviewing the increment, which they had built.

- a Sprint ends with a **Sprint Retrospective** ([4.6 - The Sprint Retrospective \(see page 126\)](#)) event, to discuss the completed sprint, evaluate learnings, plan improvements and acknowledge the team's success. The entire Scrum Team participates in the event and the Scrum Master facilitates it, while also being a part of the team. The Sprint Retrospective should last **1,5 hours** for a team, while it may be a little less for smaller teams, **but no less than 1 hour**.
- while a **backlog refinement** ([4.2 - The Backlog Refinement \(see page 113\)](#)) is not a formal Scrum event, it is a highly anticipated activity that should be conducted by the Developers and the Product Owner throughout the sprint. It is recommended that approximately **3-4 hours** are dedicated to refining the Product Backlog but no more than 10% of the team's overall capacity. Despite this being a continuous activity, it is recommended to have 2 refinement meetings throughout a 2-week sprint by default. This may be optimized by the team as they see fit with growing maturity.

4.1.3 Small Scale Scrum



In the Small Scale Scrum, we are putting the emphasis on swiftness. The sprint shift events (Sprint Review, Sprint Retrospective, and Sprint Planning) can be combined into a single meeting where all the stages are present. According to the framework it shouldn't take more than 90 minutes. In the Sprint run, there is no need to meet at the Daily Scrum because the work in such small teams can be calibrated in the run. All events are shorter due to a low number of participants. There is no default facilitator of the events and this activity usually is rotating.

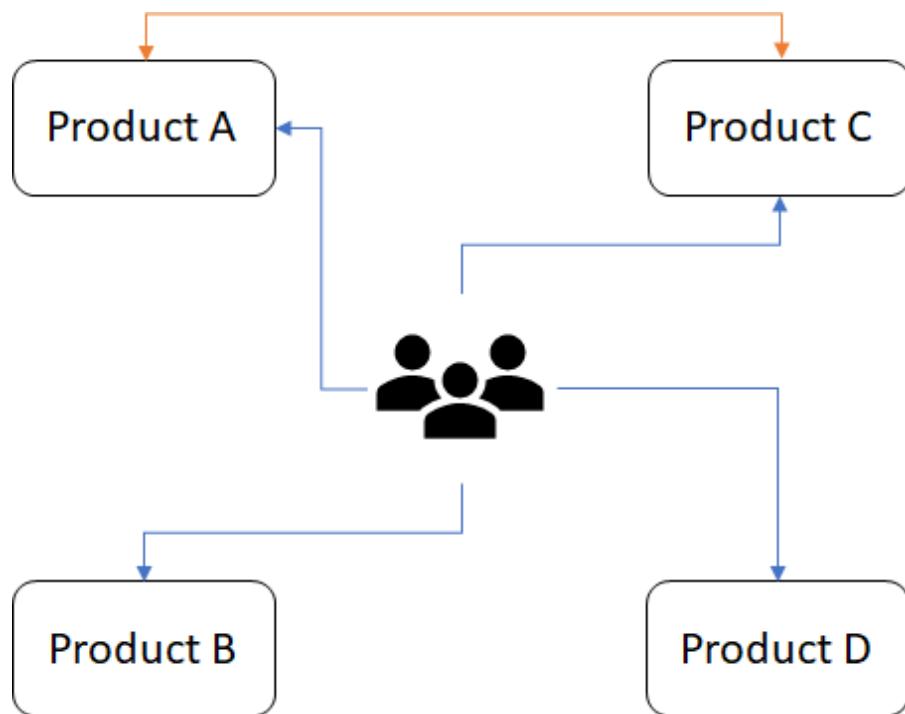
4.1.4 KanBan



Unlike in Scrum-based approaches, KanBan does not include the notion of a sprint; however, one can and should define a cyclic cadence that would represent the necessary feedback loop frequency around the Service Delivery. There are quite a few analogies of the KanBan ceremonies to the Scrum Events which are also compared in more detail in this chapter: [N2 - 4.10 - Events for Kanban teams - Next Generation Agile - Confluence EG A/S¹²](#). Most of the KanBan cadences will be unique in their own way, depending on the necessity and frequency of the acknowledged events. The cadence definition for KanBan is much looser than that of EG Scrum, however, it should still be taken into account to attempt to align the cadence synchronization between teams and products to the extent possible, independent of the selected agile delivery approach.

The events in KanBan should reflect/replace currently occurring meetings in order to avoid duplicating events that need to take place in the process.

4.1.5 High-product-granularity



The cadence for high-product-granularity teams remains aligned with the selected framework, whether it's EG Scrum or KanBan, however may recommend several alterations to the default events that provide the team with means to optimize their split focus on product delivery, while cultivating the team responsibility culture.

¹² <https://confluence.eg.dk/display/NG/N2++4.10++Events+for+Kanban+teams>

4.2 4.2 - The Backlog Refinement

While Product Backlog refinement is not a formal Scrum event, it is a crucial and extremely relevant activity in the team Sprint cycle.

4.2.1 The participants

- Developers including software engineers, testers, business analysts and UX designers
- The Product Owner
- (optional) The Scrum Master to facilitate the meeting
- (optional) Any area/domain specialists, architects as needed by the Developers or Product Owner to help the Developers understand the scope and goal of a Product Backlog item

4.2.2 The goal

The purpose and goal of the Backlog Refinement are to make the Developers understand the priority, the context, the content, and the idea behind Product Backlog items. This is done by:

- discussing sequential Product Backlog items for team's understanding and ability to estimate ([5.2 - Refining and estimating an initial Product Backlog \(see page 152\)](#)) with respect to the Definition of Ready ([5.5 - The Definition of Ready \(see page 176\)](#))
- decomposing or splitting items which are too big or too complex
- revising and confirming the prioritization of the Product Backlog items with respect to the upcoming sprints
- identifying external dependencies, risks, blockers and required spikes ([5.3 - The Product Backlog \(see page 160\)](#)) in order to assess, clarify and own them
- identifying areas for reuse of existing components/services or where created components/services could be beneficial to other teams (more in [Shared Components¹³](#))
- analyzing and evaluating incoming ServiceNow issues to gain clarity and/or determine their relevance to the Product Backlog

Therefore before a Product Backlog item is considered to be "Ready for Development" ([7.1.7 - Jira issues workflow \(see page 279\)](#)) it should be estimated according to team's best knowledge and deemed compliant with the Definition of Ready by the team. Then it can be considered for the issue to be a Sprint candidate; this can continue in a major part throughout the refinements but can last up to and including the Sprint Planning event ([4.3 - The Sprint Planning \(see page 117\)](#)).

¹³ <https://confluence.eg.dk/display/NG/Shared+Components>

4.2.2.1 Recommended level of Product Backlog items' readiness for upcoming sprints

Next Sprint - mandatory

- 95% of PB items should comply with the *Definition of Ready*

Sprint + 1 – highly recommended

- 70% of PB items should comply with the *Definition of Ready*

Sprint + 2 – recommended

- 50% of PB items should comply with the *Definition of Ready*

4.2.3 The timing

The amount of time required for a Scrum Team to refine Product Backlog items sufficiently enough to start working on them in a Sprint depends on multiple factors, such as technological or functional complexity, team maturity and seniority, etc. The needed amount of time should be ultimately established by the Developers and the Product Owner, in such a way that it allows the team to understand the work to start doing it; naturally leaving some level of uncertainty as more information may be discovered, new tasks may arise and new learning can occur during the sprint.

It is recommended for teams to start refining the Product Backlog dedicating **between 3 and 4 hours split into 2 meetings** per (2 week) Sprint. This is the time where the focus of the team should solely be on the Product Backlog items. By natural means, backlog refinement occurs constantly, throughout everyday work, that allows new information or learnings help better understand the Product Backlog items planned for next sprints. That is perfectly fine and it should not impact the amount of time planned for dedicated Backlog Refinement meetings. As the team matures, it will figure out on its own how much time and what frequency is best needed for them to prepare the Product Backlog items for work in the upcoming sprints.

4.2.4 The best practice

A few best practices, recommendations of what to do and examples of what to avoid during backlog refinement:

- don't wait till the end of the Sprint to start refining Product Backlog items for the next sprint - this is very risky may lead to team starvation from not having a sufficient number of items to work on in the next Sprint, always try to have some *Ready* items in the Product Backlog
- don't refine too early either - focus on the next 3 sprints at a maximum in order to avoid having to come back and refine the item again in the future, due to or forgotten outdated discussion
- make notes - it can easily be confusing when discussing multiple items at a time, often with a need to come back to the discussion on an item; write down what is missing from the item to be *Ready*, what was agreed, what are the risks, dependencies, etc.
- don't assume, be fully transparent - if something is required by the item to be *Ready* then it can be confirmed only when the condition is fulfilled; for example: when expecting UI designs, do not confirm adherence to the Definition of "Ready" until it has been delivered even despite promises to be so
- be strict but also understanding - remember that the Developers and the Product Owner are a part of the same team, with the same goal. Rather than questioning and expecting something to be ensured or delivered by either role, provide a helping hand, ask how to support in meeting a specific requirement; collaborate and communicate.

4.2.5 Small Scale Scrum

Small Scale Scrum is a framework that emphasizes communication between team members. Refinement, in this case, is optional. In general, we can use an approach from the Scrum framework but with one exception which is time.

Timing can be limited to 30 min per week.

Who is needed in Small Scale Scrum:

- Developers (mandatory)
- Client (optional)
- SMI (subject matter expert) (Optional)
- Representants from the Customer side (Optional)

4.2.6 High granularity teams

From a High granularity team's perspective (or in a case when a lot of domain specialization comes into place), a refinement session is a good opportunity for sharing knowledge among developers. Remember that the refinement process needs engagement from the Scrum Team perspective as a group, not particular individuals in their scope. Understanding user stories (business problem to solve) and contributing to the design of implementation method, should be all team member's efforts. Remember to keep a balance

between specialization and cross-functionality of developers in the team. Try to avoid closure in domain silos. Share your knowledge and be open to new learnings.

For teams with domain/functionality silos highly present, a refinement session might be challenging at the beginning. Becoming a cross-functional team with a common goal(s) is a process. In those cases, it's reasonable to consider splitting the refinement session into two parts:

- The common session, for all scrum team members, with a high-level perspective of functionalities and priorities
- Subgroup session(s), for developers interested in specific domain issues. Detailed level perspective, including working out the best method of implementation and estimation of complexity. Product Owner is not required here but he could be engaged if such a need arises (i.e. need for the clarification of acceptance criteria). It is recommended that the subgroup sessions are not limited only to one developer, because in such a situation we miss the knowledge share on a detailed level. It is good to align the subgroup session(s) with PO and between each other so the PO becomes available to the team for required alignment.

That approach is a good starting point and as the maturity of the team increases with experience and knowledge sharing thrive, the relation of common sessions to domain sessions should also grow accordingly. The team should strive to achieve the level of cross-functionality that enabling a swarming effect.

4.2.7 Consultancy teams

In the consultancy teams, we can distinguish two types of issues that are subject to the refinement process:

- customer-funded development which previously was the subject of review and estimation;
- other issues that are being refined in an ordinary way, which were not subject to the previous review by developers.

The refinement process to the first group of the issues should focus **on discussing and agreeing on the areas that were not yet covered in the already specified requirements**. It is due to make clear of what is expected to be delivered. On the other hand, we should **discuss the way that the scope of the work will be delivered**. We should as well dedicate some effort to the work breakdown so it is possible to deliver the important issues (customer-funded development) in the most efficient way. The second group of requirements should be refined in a standard way, as described above in the chapter.

In the consultancy teams, the Consultants are the persons who are closely working with the customers and who are highly aware of the customer's requirements and expectations. Due to this reason, it is recommended to invite the selected consultants to the dedicated parts of the Refinement meeting so they can act as domain experts and can provide the needed or required input. On the other hand, if some input or clarification is still missing, the required content can be gathered faster if Consultants are aware of that and engaged in the process itself.

If there are many issues that consultants should participate in during the Refinement meeting, it is worth reflecting to divide the Refinement meeting into two separate parts, as described in the high granularity teams above. This can improve the efficiency of the Refinement process and reduce the Consultant's engagement.

4.3 4.3 - The Sprint Planning

Every sprint begins with a Sprint Planning event, where the Scrum Team collaborates together to create a plan for the duration of the next sprint cadence.

4.3.1 The participants

- The Developers including Software Engineers, Testers, Business Analysts and UX designers
- The Product Owner
- The Scrum Master to facilitate the event
- (optional) Any area/domain specialists, architects ad needed by the Developers or Product Owner to help the Developers understand the scope and goal of a Product Backlog item

4.3.2 The goal

The purpose of the Sprint Planning is for the team to craft an estimated plan of work for the sprint, where a potentially releasable Product Increment can be delivered at the end as an outcome. Once achieved, such a plan should be reflected in Jira (see: [7.1.2 - Planning and starting a Sprint \(see page 254\)](#)). The Sprint should have a Sprint Goal that relates to the Product Goal. The Sprint Planning event can be distinguished into 3 parts: a proposition of how the product can increase in value, deciding on what needs to be done and how it could be done (task decomposition or task break-down).

4.3.2.1 Part 1 - Why is this Sprint valuable?

The Product Owner proposes how the product could increase its value and utility in the current Sprint. The whole Scrum Team then collaborates to define a Sprint Goal that communicates why the Sprint is valuable to stakeholders. The Sprint Goal must be finalized prior to the end of Sprint Planning.

4.3.2.2 Part 2 - What can be Done this Sprint?

- identifying and establishing the Sprint Goal (see below) by the Product Owner
- discussing which Product Backlog items need to be done in order to achieve the Sprint Goal by the Scrum Team
- estimating (or confirming estimates) (see [5.2 - Refining and estimating an initial Product Backlog \(see page 152\)](#)) Product Backlog items that need to be completed and checking the Definition of “Ready”
- verifying the plan against the team’s capacity and re-evaluating the plan and sprint goal if needed

4.3.2.3 Part 3 - How will the chosen work get done?

- identifying small pieces of work in the scope of the Product Backlog items, called sub-tasks (each sub-task should be assumed possible to be realized by a single person in less than a day)
- identifying a plan for the work to be accomplished during the Sprint by Developers

- confirming the plan for the sprint by conducting a sanity check, where the sum of sub-tasks delivery time is verified against the team's capacity
- presenting the plan to the Product Owner

4.3.2.4 What does a Product Goal look like?

The Product Goal describes a future state of the product which can serve as a target for the Scrum Team to plan against. The Product Goal is in the Product Backlog. The rest of the Product Backlog emerges to define "what" will fulfill the Product Goal.

(...)

The Product Goal is the long-term objective for the Scrum Team. They must fulfill (or abandon) one objective before taking on the next.

- the Scrum Guide

The Product Goal can be understood as the North Star to which the product advances in the long-term. It should be crafted by the Product Manager(s) in co-operation with Product Owner(s). The Product Goal should be reflected in the Product Backlog. The increments that the Scrum Team delivers each Sprint should contribute to the Product Goal as well as discussed during [the Sprint Review](#) (see page 122).

Examples:

- increase user retention by 15% in Q1-2
- improve customer satisfaction on NPS score from -20 to 50

4.3.2.5 What does a Sprint Goal look like?

A Sprint Goal should clearly define what is the aim and expectation to be achieved for the Product at the end of the Sprint. It should be precise enough to lead the Developers throughout the sprint, towards the result anticipated by the Product Owner and at the same time flexible enough to allow the team to self-manage with the intention to fulfill the established Sprint Goal.

The Sprint Goal should be a single objective of the Sprint; a distributed Sprint Goal leads to loss of focus by the Developers. Having a single objective Sprint Goal is challenging especially for "high granularity product" development teams. In those cases, a set of objectives is an acceptable compromise and indicator for improvement. As the team matures, the aim should be to formulate the Sprint Goal in the form of a single objective.

For example: *Providing log-in functionality to the system for registered users.*

The goal is clear, yet does not imply the "how" and allows for multiples Product Backlog items to be included in the scope of the Sprint Backlog, some of which strictly relate to reaching the Sprint Goal and other which can be deemed as optional and are negotiable if necessary.

4.3.2.6 Pre-requisites to a successful Sprint Planning

- an ordered, prioritized Product Backlog (at least top part) reflecting the objective of the Product Owner
- initially refined and estimated Product Backlog (at least top part)
- team velocity and capacity (see [5.7 - Using team metrics \(see page 184\)](#))
- past Product Increment, representing the current state of *Done* work

Preparation for the Sprint Planning event is especially crucial for teams with high-product-granularity or working with multiple products, where having a properly refined and ordered Product Backlog is key to an effective meeting with all of the participants. It is clear that with silos within the team, a long common Sprint Planning can be troublesome at the beginning, therefore proper preparation needs to take place as part of [4.2 - The Backlog Refinement \(see page 113\)](#) so that the team can focus on the common goal in the meeting, rather than inspecting individual items on a fundamental level.

4.3.3 The timing

The amount of time required by the team depends in a lot of ways on their maturity, state of Product Backlog and level of professional experience. With time, the team is likely to conduct the Sprint Planning events faster, as they improve on their own efficiency and also as a result of a better state of the Product Backlog (resulting from ongoing, continuous refinements).

The recommended time for a Sprint Planning in a 2-week Sprint cadence is **no more than 4 hours but not less than 2 hours**. While at the beginning, the value should be rather between 3 and 4 hours, with time it should be possible to limit this time to a 3h timebox. During this time, the Scrum Team should collaborate to craft a Sprint Goal and a Sprint Backlog which will allow to fulfil the established goal. In the end, the work should be decomposed into sub-tasks and there should exist a plan, both valid at least for the beginning of the sprint

It is best if the Sprint Planning event takes place at the same time and in the same place, every iteration.

4.3.4 The best practice

- try to have an idea about the rough plan ahead - it is very helpful to discuss the roadmap overview for 2-3 sprints ahead throughout the sprint (refinements) - having done this the Developers should be better prepared when coming into the Sprint Planning instead of not knowing what to expect; this may also shorten the time needed for the event
- do not make drastic changes to the top of the Product Backlog right before the Sprint Planning - the key to good planning is to have the items on top of the Product Backlog stable and detailed, while the stuff on the bottom is general and prone to change; making last-minute changes to the next sprint candidates, introduces risk for the items not to be refined well enough without enough focus from the team, this impeding their ability to forecast the work to be done
- start task break-down (decomposition) from the top-most prioritized items in the Sprint Backlog - in case you run out of time, you will have coverage for the first, most relevant user stories
- be aware of your (average) velocity and know your teams' capacity for the Sprint to be planned - you will be able to plan more accurately

4.3.5 Small Scale Scrum

The Sprint Planning meeting is time-boxed (i.e., set in advance for a specified amount of time) and focused on planning work for the upcoming Sprint. The meeting is run by the development team and advance planning is required to keep it concise. Sprint Planning is value-based.

Mandatory and optional participants of the Sprint Planning:

- Developers (Must be)
- Client (Optional)
- SMI (subject matter expert) (Optional)
- Client representant (Optional)
- Scrum Master (Optional)
- Product Owner (Optional)

At this point, requirements that will be worked on in the upcoming Sprint should be clear and contain approved acceptance criteria. Capacity is not used; instead, the team has to take an educated guess. Velocity only emerges after three or more Sprints, which is usually after the Small Scale Scrum projects are finished.

4.4 4.4 - The Daily Scrum

The Daily Scrum is a short, yet extremely relevant meeting for inspecting and adapting ongoing work as it progresses toward the Sprint Goal.

4.4.1 The participants

- The Developers including software engineers, testers, business analysts and UX designers
- (optional but recommended) Scrum Master to facilitate the event if needed, otherwise as a listener
- (optional but recommended) Product Owner as a listener primarily, for the Developers' convenience

4.4.2 The goal

The Daily Scrum is primarily an event for the Developers. It helps them maintain proper and regular communication with a focus on delivering the Sprint Goal. While other roles, such as Scrum Master and Product Owner may attend, they should not disturb the conduct of the meeting. During the Daily Scrum, the Developers check their work in the scope of the Sprint Backlog to validate if it continuously allows reaching the agreed Sprint Goal. If any deviations from the goal are identified, they should be reacted upon and the work of the team should be accordingly adjusted to "get back on track". During the event, the team raises impediments which affect or block their work, to the Scrum Master. It is also advised that the team tries to tackle the impediments themselves first. As "side effect" while participating as a listener, the Product Owner gets an overview of the progress of the work and can acknowledge the validity of the Sprint Goal. There are several elements for this event that may be worth discussing:

- discussing what was completed on the previous day by the Developers, that helped them to get closer to reaching the Sprint Goal
- discussing a plan for the day, including any alterations to the initial plan, that are necessary in order to make progress toward the Sprint Goal
- raising issues and identifying what areas of the team's work are affected, how severe is the impact and discussing how they can be resolved by the team
 - identifying issues as impediments if not resolvable by the team and notifying the Scrum Master, to support the team
- verifying overall Sprint progress with respect to the Sprint Goal; checking if the goal is still reachable and if it remains valid
- making sure that the *scrum board* in Jira is updated (should be so before the Daily Scrum) and updating the board if necessary to reflect the most actual state of the sprint work

4.4.2.1 How does it look like?

The *Scrum Guide* once suggested 3 basic questions as a standard agenda for each Daily Scrum but it no longer advocates that; the Developers are welcome to have their own agenda based on their experience and knowledge of what helps the team best to inspect and adapt their daily work. However, it is best for inexperienced teams and those with a low team-maturity to follow the common agenda:

- What did I do yesterday that helped Developers meet the Sprint Goal?
- What will I do today to help Developers meet the Sprint Goal?
- Do I see any impediment that prevents me or Developers from meeting the Sprint Goal?

The Developers can select whatever structure and techniques they want, as long as their Daily Scrum focuses on progress toward the Sprint Goal and produces an actionable plan for the next day of work. This creates focus and improves self-management. Daily Scrums improve communications, identify impediments, promote quick decision-making, and consequently eliminate the need for other meetings.

- Scrum Guide

4.4.3 The timing

Every Daily Scrum should fit within a **15-minute** time box. This is a sufficient amount of time for a properly sized Developers to reflect and plan their daily work. While it is understandable that new teams may require more time to get a hang of it, deviating more than 5 minutes from the time box signifies that the purpose of the event may have been altered and should be verified or consulted with the support from the Scrum Master or Agile Coach.

Since one of the goals of the Daily Scrum is to plan a day's work, it is highly recommended to hold the event in the morning, at the beginning of the day. Similarly to other events, it is recommended to keep this time and the location constant over a long period of time.

4.4.4 The best practice

- conduct the meeting standing up - the Daily Scrum is often informally called the daily stand-up because of this recommendation; standing up while discussing keeps the focus on the matter, reduces passive participation and helps to fit within the time box
- have the daily... every day - even if you feel that it is unnecessary because you sit and collaborate closely with your team, it is a great synchronization technique
- grab a coffee - it's a great idea to connect the Daily Scrum with a morning cup of coffee to kick-start your day!
- if you need someone specific to attend in regard to your work plans, just ask - it's 15 minutes or less, so it shouldn't be a problem for anyone. If you know this sooner, of course, don't wait and invite your "guest" as soon as possible.

4.4.5 Small Scale Scrum

Small Scale Scrum is a framework that emphasizes communication between team members that's why Daily Scrum as an event is optional and up to teams decision.

If the team members still want to have Daily Scrum they can use the formula from the Scrum framework which is 15 min meeting to check how the sprint is going at the setup plan for 24h.

4.4.6 Consultancy teams

In the consultancy teams, where the consultants are partially playing the role in Scrum setup (as Developers), they participate in the Daily Scrum as same as other team members. In the setup where the consultants don't play any Scrum role, there is no requirement to participate in the Daily Scrum by Consultants. On the other hand, if the Scrum team members are closely collaborating with Consultants and there is a strong need for information flow about current Scrum progress and calibration, the Consultants representatives can be invited to Daily Scrum meetings as observers. It is on Scrum Master's shoulders to facilitate the event accordingly.

4.5 4.5 - The Sprint Review

The Sprint Review is an opportunity for Stakeholders to review the “Done” Product Increment together with the Scrum Team and decide on how to proceed from there, in order to maximize product value. It's an important step in the process, representing all 3 pillars of Scrum.

4.5.1 The participants

- the entire **Scrum Team**, meaning:
 - The Developers including Software Engineers, Testers, Business Analysts and UX designers

- The Scrum Master to facilitate the meeting
- The Product Owner
- The Product Manager
- key stakeholders - invited by the Product Owner, having a relevant interest in the Product (this could be: customers, end-users, sponsors, etc.)

4.5.2 The goal

The Sprint Review has a very broad purpose with multiple elements contributing to its value. Initially, the aim is to summarize the past sprint and reflect that fact in Jira [Jira - Closing a sprint \(see page 266\)](#), concluded with a potentially releasable Product Increment which should be inspected through collaboration with all participants (especially the stakeholders). It should also be inspected in comparison to the overarching Product Goal. The Sprint Review is a crucial event in terms of feedback to the Product which is being built, as it can lead to change in direction of Product evolution and alterations in the Product Backlog. All this is achieved by:

- summarizing the sprint
 - the Product Owner presents what was and what wasn't Done in the completed sprint
 - the Product Owner verifies & confirms whether the Sprint Goal was achieved or not
 - the Developers discuss briefly their successes and challenges, as well as if and how they were overcome
 - the Scrum Team may present some metrics which reflect the team's progress, such as velocity / avg. velocity change, capacity & forecasted capacity, etc.
- presenting a potentially releasable Product Increment - Developers show a demo of the working software, which in rare cases can be supported by additional materials (however the focus should always be on presenting real working product functionality that was delivered in the last sprint); validating user stories' acceptance criteria as they are reflected in the Product Increment
- discussing and gathering feedback
 - the Developers answer questions from the participants and engages in discussion, which could lead to maximizing the Product value
 - all participants exchange feedback and provide input for the Product Owner and Product Manager to reflect in the needed changes in the Product Backlog
- making decisions and planning next steps
 - verifying the Product development progress against the Product roadmap
 - verifying the business case validity considering the updated timeline and market conditions
 - verifying high-level priorities with respect to the budget, timeline, etc.
 - deciding on release dates, go / no-go decisions

4.5.3 Preparing a good Demo

A good demo should briefly and efficiently present the delivered, working and "Done" functionality, without excessively taking up the time to run through all scenarios, use cases and features of the Product. It should present that the Sprint Backlog items have been completed with respect to their Acceptance Criteria and keeping in mind the Definition of "Done". The demo is extremely valuable especially to teams with high-

product-granularity, distinct silos and multi-product focus, where it constitutes an opportunity to share knowledge and inspect the not-so-familiar areas of the team's work. A few things to consider and remember during a Demo:

- present the real, live, working software, not a recording nor a PowerPoint presentation
- focus on the *Product Increment*, so the features that have been added on top of the existing Product - use the existing features if needed but focus on the "new" (this is not regression testing - however, it is expected that all existing features remain intact and working during the demo)
- describe how the features address the Sprint Backlog items and their Acceptance Criteria, as well as the Definition of "Done"
- if the number of items to be shown in a demo is significant, then focus on the highlights which address the Sprint Goal or combine them to be shown in a single-use case scenario - make it interesting!

4.5.4 User Acceptance Testing in Agile

Per Agile Manifesto: "*Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*" Based on that, clients and end-users should be engaged as early as possible in the delivery process to satisfy their needs, especially in customer-oriented organizations as EG.

When not integrating UAT early in software development cycle, we may expect challenges such as not fulfilling original client's request due to late validation, missed edge cases that result in rework, clients having not enough time for UAT testing and struggling to perform them.

In Agile teams, the Product Owner has the responsibility of maximizing the value of the product, and represents all stakeholders, including customers and users. The Product Owner is also an authorized entity for approving User Acceptance Testing.

For mature Agile teams, with strong Product Owner role as a customer representative, following practices are advised:

- UAT is a part of every sprint, which provides opportunity for active learning and correcting errors at early stages
- Stakeholders accept the expectations of including UAT into Agile process and understand its benefits
- UAT starts when user story is written and Acceptance Criteria are created
- Business, development and testing group is involved in every sprint. They can co-create Test Plans and Test Cases
- A test management platform is utilized to make developing tests, recording results, and communicating with testers easy and personalized
- If there are sufficient competencies and tools in the team, acceptance test-driven development (ATDD) or behavior-driven development (BDD) is recommended to comply with fully Agile approach. In addition, EG test management tool supports BDD approach.
- UAT can be done in multiple places: throughout the Sprint/cadence and during Sprint Review or Demo. Usually, the feedback from stakeholders is collected and processed during the Sprint Review meeting
- The Product Owner is finally responsible to accept the deliverables in the sprint and approving the User Acceptance Tests.

Ultimately, UAT can be embedded into the entire Agile workflow. It starts at the beginning of the cycle with co-creating features and continues when having touchpoints with customers and incorporating their feedback. UAT should not be perceived as the one of the last and separate steps before finishing the delivery.

4.5.5 The timing

For a single-team, single-product Sprint Review, its duration should be **no more than 2 hours but no less than 1 hour**. The amount of time needed for conducting the review properly may vary depending on the scope of the Sprint Backlog being presented as a “Done” Product Increment, as well as the amount of discussion that can take place afterwards. The Scrum Master should ensure, that everyone keeps within the timebox.

While often it is the case that multiple teams work on a single Product and a single Product Backlog with their own Sprint Backlogs (see [6.2 - Cross-team collaboration in a single product \(see page 219\)](#)), in such cases the timing of the Sprint Review needs to be adjusted accordingly. While multiple elements of the event are common for all teams (verification of the Product roadmap), some are a distinct per team and require more time overall (demo, Sprint Goal). It is recommended to **add 30 minutes** for each additional team as it joins the Sprint Review of a single product.

4.5.6 The best practice

- nothing gets done by itself - making the Sprint Review an interesting, showcase event requires preparation from the team on the demo to be done by the Developers and on the sprint summary to be done by the Product Owner; take a few minutes at the end of the sprint to get everything ready!
- make the Sprint Review a unique event, which incorporates not only inspection and adaptation of the Product Roadmap but is also an opportunity for the teams to collaborate together, to exchange information and to get to know each other better (have some integration) from time to time -consider review fairs or open spaces (see more in [6.2 - Cross-team collaboration in a single product \(see page 219\)](#)).
- very often, key stakeholders have a very busy and restricted calendar - plan the Sprint Review together with them to occur consequently at the same time and place during the sprint, so that their calendars may be aligned and always have time for this event. It is crucial to the Product for stakeholders to participate, see the results and provide feedback at least with each sprint change.
- if the team(s) is/are distributed in different locations, make an effort to bring them together in one place at least every quarter (every 6 two-week sprints) for a sprint change (including the Sprint Review) to have an on-site, co-located experience with some integration elements as well.

4.5.7 Small Scale Scrum

The Sprint Review meeting is organized and run by the development team and attended by the customer or customer team.

The Sprint Review is time-boxed and contains demonstrations of Sprint work and a short outline of work completed/not completed, bugs raised, and known and/or descoped issues (if any). Customer feedback is gathered at the end of the demonstration and incorporated into future Sprints. Very little (if any) preparation is required to keep the meeting short and structured.

The Proof of Concept/Demo is a realization of the small work completed in the Sprint to showcase progress in development. Any deviations or incomplete work are discussed during the POC/demo.

Who participates in the Sprint Review:

- Developers
- Customer
- SME (subject matter expert)
- Customer representant
- Scrum Master (Optional as this role is optional)
- Product Owner (Optional as this role is optional)

Responsibilities of inviting people to review meetings on behalf of developers. They are deciding who they want to show working increment and who will bring value to this meeting.

4.6 4.6 - The Sprint Retrospective

The Sprint Retrospective is the final event in the Sprint cadence cycle, and probably one of the most important ones, as it brings to life all 3 pillars of Scrum, fostering continuous improvement.

4.6.1 The participants

The Sprint Retrospective is for the **Scrum Team**, so:

- The Developers including software engineers, testers, business analysts and UX designers
- The Product Owner
- The Scrum Master to participate as a team member and to facilitate the event

4.6.2 The goal

The purpose of the Sprint Retrospective is to self-reflect on the functioning of the Scrum Team, with a specific focus (but not limited to it) on the recently completed Sprint. The retrospective allows the Scrum Team to review their work in a *Transparent way*, *Inspecting* all positive and improvable aspects of the team, in order to incorporate the needed *Adaptations* into their future iterations. The expected outcome of the Sprint Retrospective is (at least) 1 improvement item (but no more than 3 - to limit the focus and increase the chance for completion), to be placed into the next Sprint's Backlog ([5.4 - The Sprint Backlog \(see page 171\)](#)) and a retrospective note to be documented in Confluence ([7.2.2 - Documenting a Sprint Retrospective \(see page 321\)](#)). While the result is of significant value, the execution of the self-inspection is equally profitable to the team's well-being. This is achieved by:

- inspecting and discussing all aspects of the team's work, which are possible to be identified, such as:
 - people in and around the team, as well as interactions between them
 - processes which drive the team's work and impact the team itself
 - tools used by the team throughout their Sprint
- identifying the positive aspects of the team, their work, surrounding and the last sprint, as well as finding room for potential improvements

- selecting 1 or more of the identified improvements (deemed most valuable and most important), drafting a plan for incorporating the improvement and adding it into the Sprint Backlog
- validating improvements agreed on from the previous Sprint(s)
- reviewing Sprint/Scrum Team artefacts, such as:
 - results of the sprint
 - sprint burn-down chart
 - team velocity chart
 - team working agreement
 - Definition of “Ready” and Definition of “Done”
- conducting the meeting in an open-minded, trust-based and tolerant environment
- having fun and integrating!

4.6.2.1 5 stages of a good Sprint Retrospective

1. **Set the stage** - begin with a warm-up, an introductory exercise; this will help the entire team to switch to the context of the Sprint Retrospective, get comfortable and focus solely on it, while getting their other concerns and previous activities out of the room.
2. **Gather data** - follow through by inspecting the sprint, the team and their surroundings to identify points worth discussing; seek for not only those things which may need to be improved but also those which worked well, where the team did a good job.
3. **Generate insights** - discuss and brainstorm on the identified items, as here “sky is the limit”; try to look at them from various perspectives, what is their impact, the circumstances and the people/ artefacts involved. Identify duplicates (group them) and exchange different viewpoints.
4. **Decide what to do** - organize the list of identified items and prioritize them; select the most important, most needed and most critical ones - for example: by voting. Discuss how you plan to implement the improvement, what and who is needed.
5. **Close** - summarize the retrospective and what was agreed; conclude with a cool-down exercise to finalize the meeting with a positive impression and a takeaway for everyone. This is also a good moment to get feedback on the retrospective itself: sort of retrospective of a retrospective.

4.6.3 The timing

The recommended time for a Sprint Retrospective in a 2-week Sprint cadence is **1,5 hours**. This is the optimal amount of time required for conducting a valuable retrospective for a complete Scrum Team, as it allows to properly implement the 5 stages and leave enough room for discussion and agreement of improvements. It is acceptable for the event to be reduced to no less than 1 hour, for small teams (see [3.1 - What an EG scrum team looks like and how it operates \(see page 47\)](#)) since the time is closely related to the number of participants. The Sprint Retrospective should take place at the end of every sprint, without ever being neglected.

It is best if the Sprint Retrospective event takes place regularly at the same time, however, the place can be altered from time to time (not too often) to introduce a sense of freshness, should some feeling of

stagnation or routine creep in. For example, having the event in the outdoors every now and then during the warm months can be very beneficial to the team insights.

4.6.4 The best practice

- differentiate the event with every iteration - in order to keep the routine away, always try to do something differently: different techniques, exercises, location or even small things like different seating order, different marker colours, different snacks... the little things matter.
- focus on what is feasible to be accomplished in the next Sprint - don't overestimate, as you have Sprint work to get done as well. Select items which are most important but also manageable; if something requires more time, split it up and plan it in steps or stages.
- record all of the significant improvements/impediments which were identified in the form of a improvements or impediments list/backlog and keep them somewhere visible - like the Jira favourite filters or Jira team dashboard
- use whiteboards, flip charts, post-its, coloured markers, stickers to collaborate on-site - if not possible on-site then there are multiple tools that can be used for dispersed teams; even so, try to have an organized on-site retrospective once every quarter.
- have some coffee, drinks (non-alcoholic) and snacks - make it a fun team event!
- Irrelevant of the framework or approach, as well as how cross-functional the team is - the event should ALWAYS take place together. This is an event for the TEAM.
- Link to examples of how retrospective can be held: [8.3.1 Retrospective \(see page 427\)](#)

Small Scale Scrum

The Sprint Retrospective meeting is organized and run by the development team and may be attended by the customer and/or customer team. The Sprint Retrospective is time-boxed and requires no advance preparation. Due to the small size of the development team and the Sprint builds (with a smaller number of features delivered), this meeting is relatively short.

The Sprint Retrospective takes place after the Sprint Review and before the next Sprint Planning meeting. The Sprint Review, Sprint Retrospective, and Sprint Planning may be combined into a single meeting which we term the Sprint Termination, lasting approximately 90 minutes.

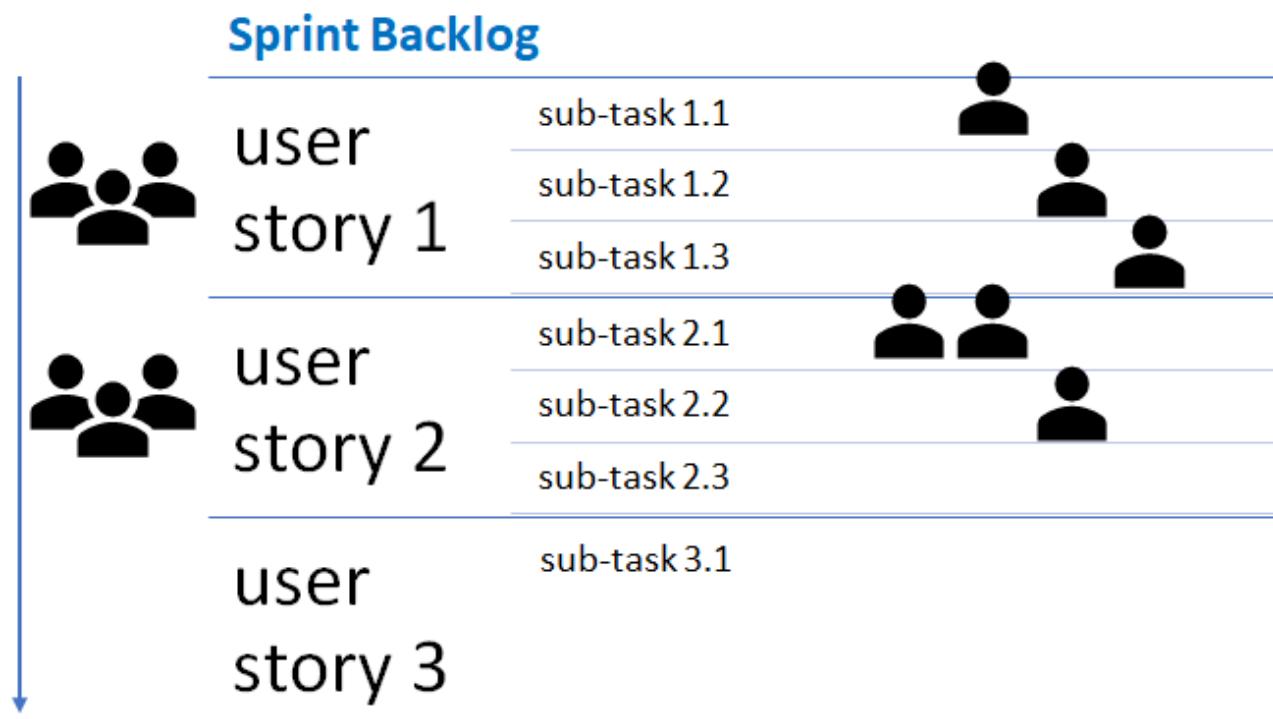
These meetings are concise, structured (thanks to advance preparation), and must not include any unnecessary/unrelated discussions.

4.7 4.7 - Daily work in a Scrum Team

Set aside all of the Scrum events, the actual work is done to deliver a potentially releasable Product Increment at the end of the sprint, takes place on a daily basis. The Developers collaborates to complete the Sprint Backlog items committed during the last Sprint Planning by creating UX designs and project documentation, by designing the architecture and by implementing/coding the features defined in the User Stories with respect to the Definition of Done. While doing so, there are a few things to consider in order to make them work as efficient and as focused on the Sprint Goal, as possible. While these practices are not limited to the following, they are highly recommended.

4.7.1 Sprint priorities

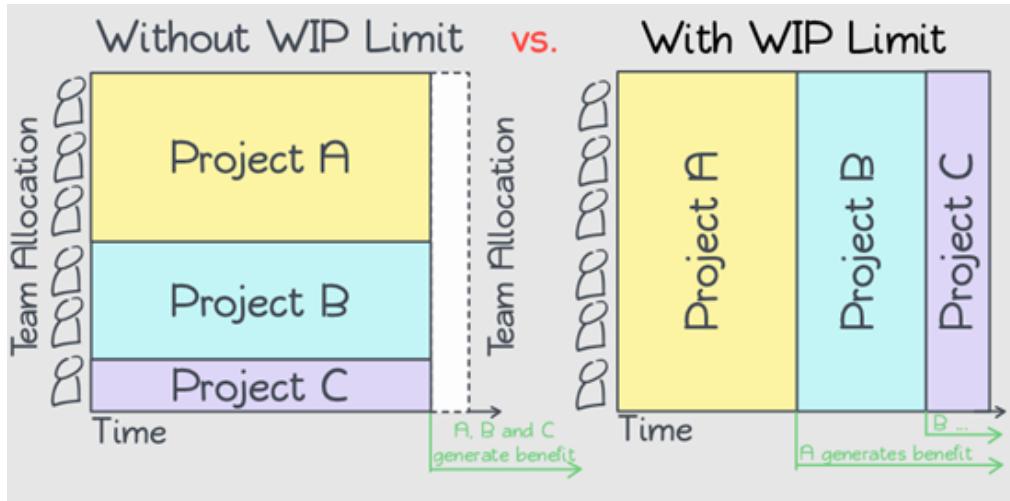
The Sprint Backlog should reflect the prioritization as taken from the Product Backlog, meaning most important items should be at the top and least important at the bottom. While that may be altered during the sprint planning, as an outcome from the team discussion, in the end, the final Sprint Backlog order should reflect the order of priorities for the team.



It is crucial to obey 3 simple rules when proceeding with work items from the Sprint Backlog:

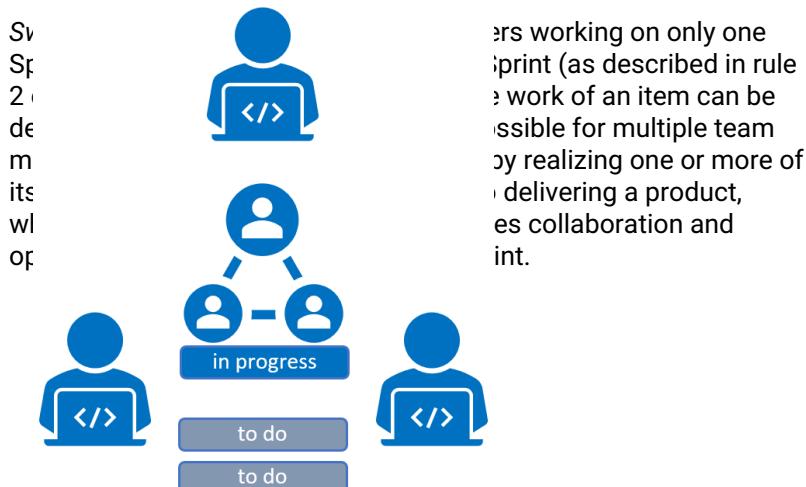
1. **Always take the priority of items into account - choose the item at the top first before proceeding with the next.** The items at the top are there for a reason - they reflect the most valuable and most desired features by the Product Owner; this should be respected. In case all forecasted work cannot be delivered, the Product Owner will prefer to end up with the most relevant items delivered. In addition, usually, the items at the top are the most connected with the Sprint Goal, hence focusing on them is crucial to following the path to achieving it.
2. **Parallelize work on the specific Sprint Backlog item as much as possible.** When Developers are working together on a specific (top priority) backlog item, trying to finish one first before moving on to the next (instead of working on multiple items at a time) they increase the chance of delivering value at the end of the sprint (even if not all Sprint scope is completed). During break-down of user stories into sub-tasks take that into account as well, as it will simplify sharing the work of a single Sprint Backlog item among several team members. When planning new work, always look first - talk with the team - if it is possible to support them in completing a higher prioritized item first before moving on to the next.

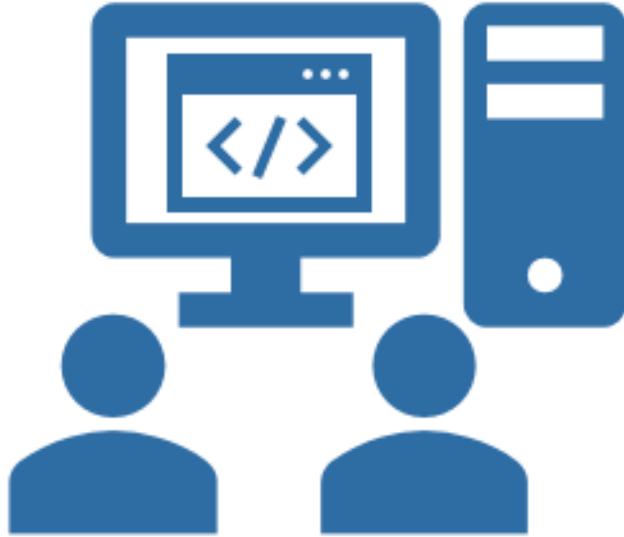
3. **Limit work in progress** - this practice will help you to avoid wastes of time and energy drain dedicated to context switching and improve the delivery rate by focusing on fewer items at a time aiming for completion. Please see the below visualization for better understanding.



⚠️ While it is not always possible to follow the above 3 rules because change is something all teams should always anticipate (considering limited specific skill set, item granularity, etc.) they should always be regarded first. If there are issues affecting those rules constantly, that the Developers is not able to deal with - please consult with your Scrum Master or Agile Coach.

4.7.2 Swarming vs. pair-programming





Pair-programming occurs when 2 (a pair) of developers work together on 1 workstation (potentially using a second one for reference, sanity checking or reviewing) to deliver a single decomposed piece of work, such as a sub-task (or even its part, like a component, module or other pieces of code). Some of the benefits of pair-programming are:

- faster problem resolution
- fewer mistakes and errors
- knowledge sharing
- team-building

While both techniques bring benefits in everyday work and foster collaboration, they differ significantly. *Swarming* should be regarded as a way to better organize the work within a Sprint to deliver the most value with the highest certainty, while *pair-programming* is an alternative way or an approach to software development often

used in combination with other Agile practices and frameworks, such as Scrum.

Both are highly recommended and desired to be used in the daily work of Developers; it's encouraged to attempt to adapt these techniques in all teams if needed with support from the team Scrum Master or Agile Coach.

4.7.3 Assignment of user stories and sub-tasks

User stories and sub-tasks in Jira should be assigned to the team member, who is doing the work on the item latest at the start of the work. During the initial planning of the Sprint work and further re-planning, assignments can also be made to support the plan of the Developers for future items.

Most of the time sub-tasks are one-person work items, hence they can be assigned to 1 responsible Developer. User stories can be shared by the team, therefore usually it's not possible to assign 1 specific person on the team (while Jira does not allow multiple assignees). In such a case, the Developers should choose the most involved or simply the representing team member who will be the designated assignee. If assignment changes to work handover, it also needs to be reflected in Jira, for example:

- 1 Developer taking over the work due to substitution, or for testing
- the Product Owner taking over the User Story for verification and acceptance

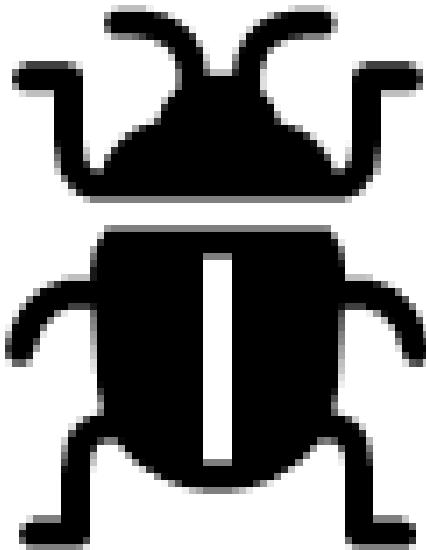


Read more about operating the Active Sprint dashboard view in Jira for daily work: [7.1.3 - Using the Active Sprint dashboard view \(see page 260\)](#)

4.7.4 Bugs

In the scope of a sprint there can be 2 types of bugs:

- those in the scope of the Sprint Backlog as planned, or as added to the sprint when necessary (see: [5.4 - The Sprint Backlog \(see page 171\)](#))
- those include issues resulting from incoming ServiceNow tickets which when assessed, may result in a bug that is added to the Product or Sprint Backlog for resolution
- those resulting from ongoing development and testing within the sprint, associate with work on a Sprint Backlog item



The first type of bugs should be treated similarly to other Product Backlog items and handled in accordance with their priority. They are usually defects discovered, originating in the production or pre-production environments. The only exception are issues those reported directly to Jira or via ServiceNow that hold a high priority or severity and they need to be handled urgently, since their relevance exceeds that of some non-Sprint Goal-related issues in the ongoing Sprint Backlog.

The second type of bugs is relevant to handle as soon as possible in order to deny further development of faulty functionality leading to defects in the future. Whether these are defects related to the functionality which is in development or regression defects related to some existing, in-production features, the rule is simple:

YOU BREAK IT - YOU FIX IT!

Taking responsibility for your own work is an important element of showing respect to the remaining team members. While Developers are accountable for the end result, it is crucial to avoid playing “the blame game” in the end. Simple enough - take responsibility for your work: if you cause a bug/defect, be responsible and respectful enough to recover the software to its working and stable state.

4.7.5 Communication and collaboration

Throughout the entire Playbook, the concepts of communication and collaboration will be raised frequently as they are fundamental factors of any proper functioning team, whether it's Scrum or other framework, techniques, methodologies.

In everyday work, these 2 concepts could be even more relevant than elsewhere and it must be in everyone's interest to maintain the communication and collaboration within the Scrum and especially Developers on a high level.

Some aspects to consider:

- discover new ways of co-operating on work items to spread the knowledge and boost quality; in addition, collaborating across the team improves the team spirit and raises morale
- choose your communication method optimally:
 - Direct communication above all - if possible, is most effective - talk to each other!
 - Video calls (MS Teams) is next when direct communication is not possible
 - Audio / phone calls
 - Instant Messaging (MS Teams)
 - E-Mail
- make sure to properly escalate issues & impediments within your work
 - try to resolve issues on your own (usually a challenge in your work item) - if you are struggling...
 - don't wait too long to ask your team for help, as someone might be able to help you resolve it faster - in the end, the whole team is responsible for the result, so focus on the Sprint Goal, not your individual benefits
 - for any issues related to challenges in your sprint and also issues that affect your work externally, try to think with the whole Developers group on how to resolve it
 - if you see that the issue is really an impediment and it is out of the scope of the Developers capabilities to resolve it, raise the impediment with your Scrum Master or Agile Coach
 - ultimately, if the Scrum Master or Agile Coach is not able to deal with the impediment, they should contact your Manager for support
- remain open-minded for opinions, ideas and experience of your team colleagues and people who are trying to support your team in delivering the highest value possible for the product, i.e. sharing components/services across EG teams (more [Shared Components¹⁴](#))
- make your Scrum Team aware of any special circumstances which may impact the Sprint or the Sprint Goal, such as absences, identified risks and impediments; the sooner they know, the better
- if your work depends on information/data which is external to the Scrum Team (such as ServiceNow support, Backstage shared components) ensure proper collaboration through means of direct contact and via Jira issue comments for the related cases

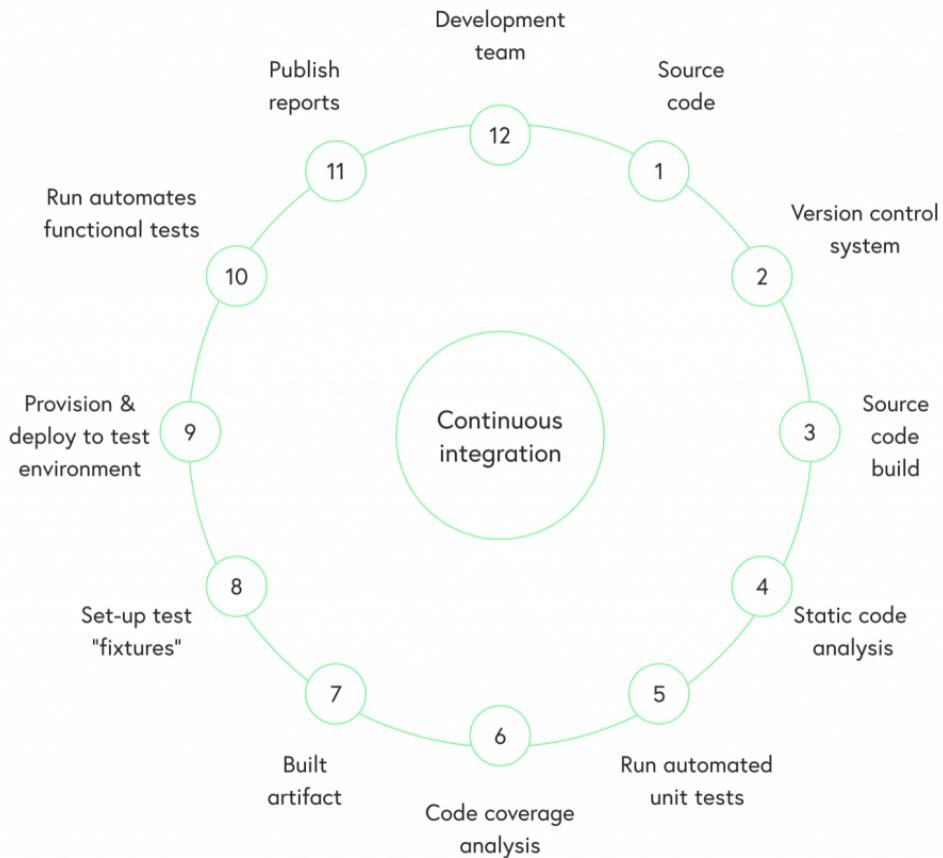
4.7.6 Daily DevOps

4.7.6.1 Continuous Integration

Daily developer's job in EG makes heavy use of Continuous Integration. It's the practice of merging all developers' working copies to shared mainline as often as possible, even several times a day. This practice

¹⁴ <https://confluence.eg.dk/display/NG/Shared+Components>

encourages committing small changes more often over committing large changes infrequently. Each commit (and push to the remote repository) will trigger an automated build process on Jenkins. During the build process, automated tests are run that help to identify if anything was broken by the changes. Continuous Integration approach and frequent commits often give you the fast feedback loop (if your change breaks anything, you will get feedback from Jenkins as soon as possible) and increase transparency and visibility (the entire team knows what's going on with the builds as well as get the latest results of tests, so they can raise issues and plan their work in context).



4.7.6.2 Branching models

The two branching models recommended to work with are:

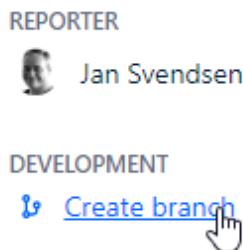
- [git-flow¹⁵](#) - a strict branching model designed around the project releases, useful for managing code in larger projects, that have a scheduled release cycle.
- [feature-branch workflow¹⁶](#), aka GitHub workflow - a lightweight, branch-based workflow that supports projects where deployments are made regularly and often.

¹⁵ <https://confluence.eg.dk/display/NG/git-flow>

¹⁶ <https://confluence.eg.dk/display/NG/feature-branch+workflow>

4.7.6.3 Creating a branch

Working on a task usually starts with the creation of a new branch in the source repository. You can do it from Jira issue view:



A branch created from within Jira will have the Jira issue key in its name. Having the story and branch connected will give you the development-related information in the Jira issue view later - like the list of branches, commits and pull-requests.

4.7.6.4 Committing files

Tip

Commit messages are important, especially since Git tracks your changes and then displays them as commits once they're pushed to the server. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.

When you finish your work and would like to merge your changes into the integration branch (master or develop, depending on your project's branching model), you need to create a pull-request.

4.7.6.5 Pull-requests

[Pull requests](#)¹⁷ let you tell others about changes you've pushed to a Git repository. Once a pull request is sent, interested parties can review the set of changes and discuss potential modifications.

Pull requests are a major communication channel at a critical junction for your code. They provide an opportunity for review and improvement before the code moves on to the next stages of testing and production, where those changes are much more difficult to back out and waste a lot more time of a lot more people. Another advantage of the pull request is knowledge sharing in the team. This is also the reason junior or less experienced developers should be invited as pull request reviewers.

¹⁷ <https://www.atlassian.com/git/tutorials/making-a-pull-request>

4.7.6.6 Pull requests serves many purposes:

- It runs automated tests and checks on the changes in your branch without affecting others.
- It allows others to see your changes, learn from them and propose improvements before they are made part of the master branch.
- It aggregates many minor commits into a single large commit to keep the history of the master branch clean.
- It serves as a tool for discussion on proposed changes. Instead of a verbose mail, just do the change and do the discussion in the PR.
- It is a quality gate that makes sure junior developers also deliver senior level code.

4.7.6.7 Do not add reviewers before:

- The PR is small and focused enough to be easy to review, split into several branches if needed.
- The build is green.
- There are sufficient tests added.
- You have reviewed (and potentially improved) the changes.
- You have written a title that describes the changes well (or just include a well-named branch name in the title)
- The description describes what your intentions with the PR is, files to ignore/skim, files to focus on.

4.7.6.8 Reviewing a pull request

- Who should review? Someone else on your team that has experience in the product, but sometimes you can add more to let people know about significant changes. Also include juniors if it gives them insights into the product.
- Once reviewers are added, give them enough time to review it. Expect at least 4 working hours of wait time, up to 8. If nobody (or not the right persons) have reviewed by 8 work hours, nudge them to remind them. So once a PR is ready for review, start other work while waiting. Remember that the first thing you should do after sending a PR out for review is reviewing others PRs.

4.7.6.9 What to look for in a review?

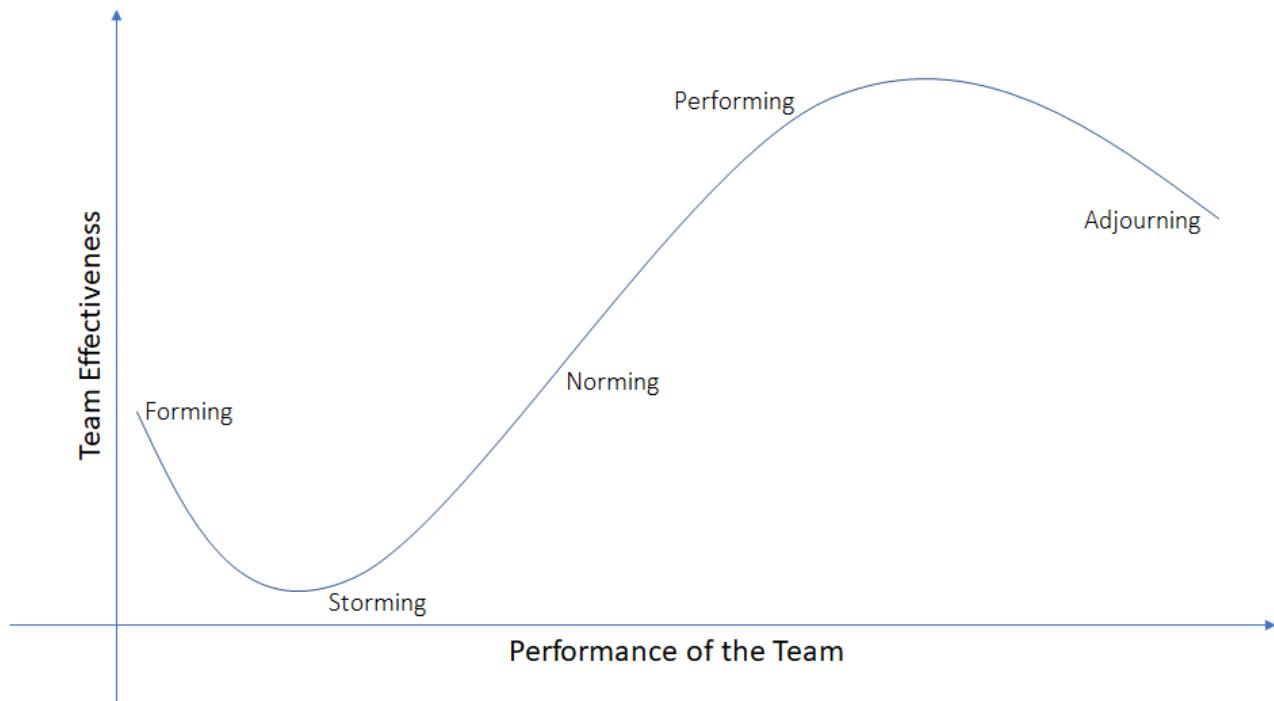
- Are the correct tests added
- Obvious mistakes in logic
- Performance/security issues
- Fundamental changes in the product

4.8 4.8 - Team building

Whether it's building a brand new Scrum Team from scratch or sustaining and evolving an existing one, team building practices are essential to obtain a stable, long-living team. The approach and techniques mentioned

further apply to both: teams, which need support in ramping-up or teams, which want to achieve the next level of efficiency and productivity.

4.8.1 5 stages of team development



Bruce Tuckman's Model of Group Development

According to Bruce Tuckman, there can be identified 5 distinct stages of team building. Each particular stage represents a stage in the team's progress towards reaching an optimal performance level and each stage characterizes with specific observed team behavior:

- **Forming** - this is the initial stage, usually the beginning period of a new team, where everyone is anticipating to get to know each other and the tasks they will be conducting as a team. The members are not yet acting as a team, but rather a group of individuals. It may take some time for them to work together before moving to the next stage.
- **Storming** - this is the stage, at which the boundaries are being tested and team member's individual habits clash against each other, often leading to conflicts and frustration. During this stage, relationships and distinct roles clarify within the team, as well as is the goal of the team, which may be challenged and questioned. Many teams do not last through this stage and dissolve.
- **Norming** - once past the Storming stage, the team begins to stabilize. The rules and boundaries have become set, relationships have been established, the team begins to identify each other's strengths and weaknesses. Members of the team associate themselves with the team identity and commit to the goal. This stage may overlap in cycles with Storming, if significant changes to the team are introduced, in order to adapt to them.
- **Performing** - this is the most anticipated stage of the team's development, since at this point their efficiency has reached the optimal level. The team is no longer focused on establishing its rules, processes and relations, hence strives to reach its goal and purpose with highest performance. Minor

personal changes to the team at this stage, should not have a significant impact on the overall efficiency.

- **Adjourning** - this is the last stage, also added by Bruce Tuckman to the model some time later after defining the initial 5 stages. It represents the moment when the team dissolves, either due to completion of its purpose and goal, or resulting from external factors, such as intentional restructuring. Some teams do not reach this stage for many years, if their intent is to be formed as long-living teams dedicated to sustainable products.

4.8.2 Team election

Scrum teams can come to be in various organizational circumstances: quite frequently the set of team members is pre-defined or is formed from a pre-existing group previously conducting other tasks. In other cases, teams are formed as a result of restructuring or taken as subsets of a larger group. In the later, it is a good and beneficial practice to conduct team self-election workshops.

Some of the benefits include:

- higher team morale
- sense of responsibility and accountability for the constructed team
- shorter Storming stage, due to potential existing relationships between team members
- team self-management introduced at the very beginning stage

In order to conduct the election workshops:

- the affected group should be gathered together
- the goal / purpose of the future team(s) should be presented (usually representing a product or parts of the product that need work)
- the restrictions or requirements should be addressed, which need to be considered by the participants during team election, such as:
 - team seniority balance
 - mandatory team competences
 - other factors as seen needed for the goal and purpose of the new team(s)
- using a whiteboard or flip chart, empty sections for team(s) should be created
- using a timebox (depending on the number of participants), the participants should discuss and self-assign themselves to the team(s)
- after all participants completed the task, team rosters may be read out and confirmed (checking if all of the restrictions have been met)

4.8.3 Team identity

With both newly formed teams and with existing, but dysfunctional teams it's extremely helpful to attempt to establish a team identity which will help to bring the team together to focus on a common goal. There are 3 basic elements, which can contribute to establishing a better team identity:

1. **Team name** - every team should have a name. The name should be chosen by the team itself, by means of a vote or any other technique if there are multiple propositions made within the team. The team name cannot be imposed by anyone external to the team, such as a manager, however there can be guidelines to the name in order to reach a convention across the organization. This guideline

can specify that a team name must be unique and must be of a given type (for example, name of an island).

2. **Team logo** - a team logo supports the team identity visually. It may have similar guidelines to the team name and perhaps additionally a unified visual requirement across the organization, to strengthen the organizational identity in addition to that of the team.
3. **Team slogan** - a team slogan supports the team identity verbally. It should be simple, catchy, most likely a single simple sentence. The team slogan should relate strongly to the vision of the team, thus adding another level of building the commitment of the team towards its goal and purpose. The slogan should not be offensive or disturbing in any way and it should adhere to the organizational values.

These three elements may be established by newly formed teams also as an additional part to the team election workshops. For existing teams, they may occur at a desired moment in time - but the sooner the better.

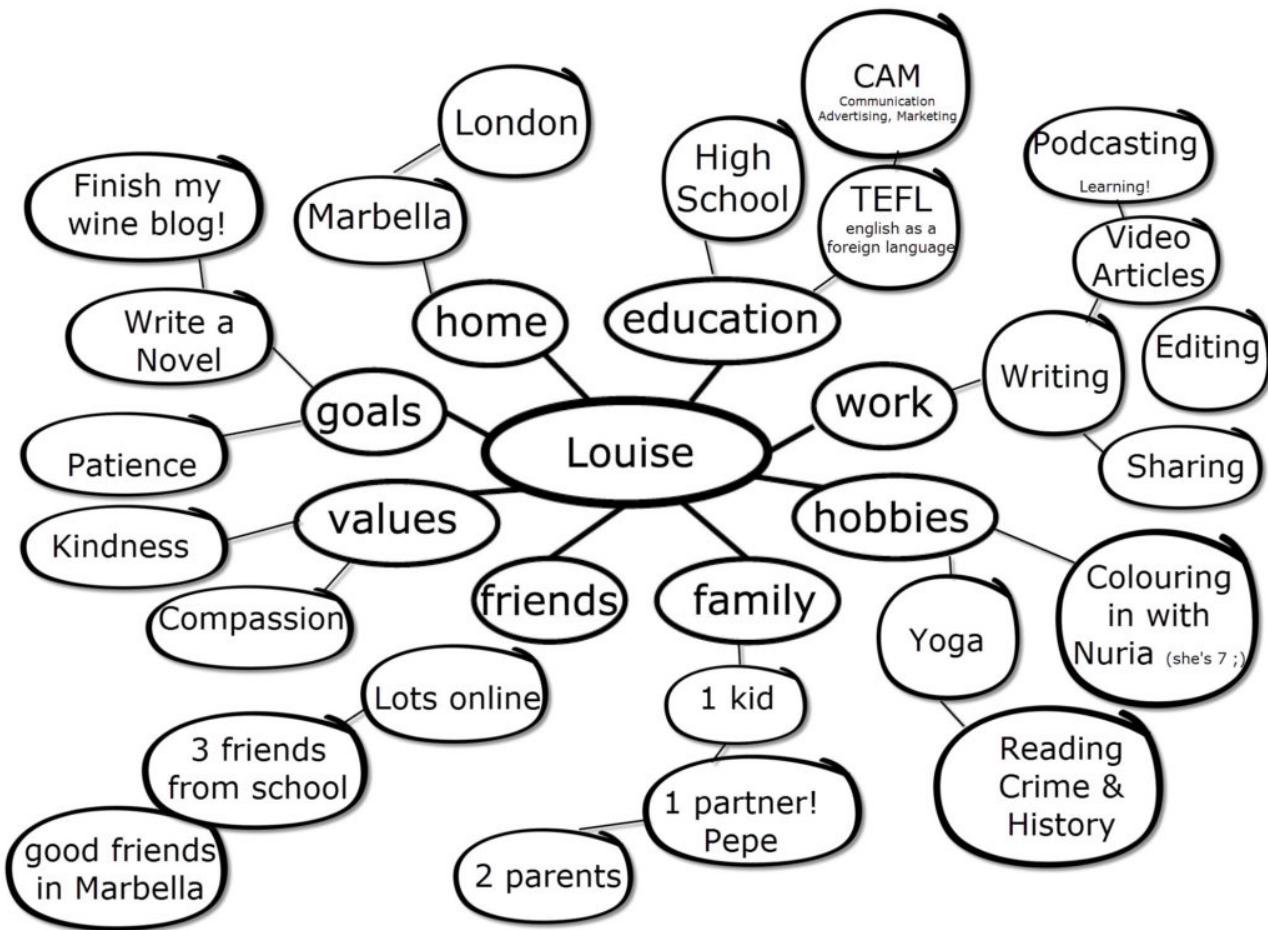
4.8.4 Team personal maps

A good way to get a team started (or restarted for that matter) is building personal maps together as an integration activity. The exercise is best conducted as a short workshop with the team - it may as well be a part of one retrospective. Personal maps are a concept from Management 3.0 and prove to be very effective in improving team collaboration.

4.8.4.1 Create your own

Gather the team for everyone to create the personal maps on their own and finally share with the rest of the team one by one. Of course set a pre-defined timebox for the exercise. To create the personal maps:

1. Write your name in the middle
2. Note down max. 10 main topics around your person - it's good to make them common across the team
3. Note details with respect to your person regarding the main topics (sub-topics can be introduced to introduce a better, more detailed structure)



Personal map example from Management 3.0

4.8.4.2 Create one for your teammate

A team that has been working some time may want to take this exercise to another level. Instead of building a personal map of yourself as an introduction, create one for a one of your teammates (sitting at a table, choose the person to your right, and so on for the rest of the team). Build his/her personal map, present and discuss how well you were able to identify your colleague.

4.8.5 Team working agreement

Another fundamental tool to use, whether it's a new team or an existing one, trying to become a bit more self-managed, is a team working agreement. A team working agreement is a set of rules created and agreed by the entire team. This document should be enforced and followed strictly by all the members of the team who participated in its creation; occasionally the TWA may be updated (every quarter or half of a year) to improve it or if there have been significant changes to the team structure. It's a good practice to hang the TWA in a visible place near the team's location or in a virtual space, such as the Confluence team page ([5.8 - Team page \(see page 211\)](#)).

The rules mentioned in the team working agreement should focus on the following categories:

- meetings: time, location
- compliance with DoR and DoD
- planning: capacity, velocity, leaves
- working with backlog, stories and bugs
- availability: Scrum Team
- team collaboration: punctuality, honesty, respect, helping others
- coding standards & code reviews

It's a good idea to prepare the first team working agreement as a workshop or as a part of one of the Sprint Retrospectives. Similarly, when in need of verifying and updating the TWA. The workshop can be conducted in many simple forms; for instance:

- gathering input from all team members, on rule and agreement propositions in any of the mentioned categories
- dot-voting on the top most 10-15 propositions overall across all categories and using them as the initial set composing the team working agreement

As the team works with respect to their TWA, they will inspect it and adapt it if needed periodically.

4.8.6 Team expansion & modification

Despite high focus on stable, long-living teams, the occasional need for changes within Scrum Teams is inevitable. Whether the cause is staff rotation or expanding the team size (more resources may be needed within the size limits of a Scrum Team) there should be an organized way to introduce the changes with minimized impact on the team performance and morale. This section focuses in general on high capacity changes, concerning multiple team members, where the need to form additional teams arises. Single-member modification should be manageable within the team itself, as long as the principles of Scrum are respected.

There are 2 main approaches to introducing team changes:

- “divide & conquer” or mixing
- shadowing

4.8.6.1 Divide & conquer

The “divide & conquer” strategy is based on taking an existing, experienced team and splitting it up into 2 new teams while adding “new members” to both of the newly formed teams. This is a more revolutionary approach, since it introduces the most changes to the current, functioning team setup. However, it will result in faster team ramp up, due to the close collaboration of mixed experienced and new team members. The drawback is initial drop in team velocity and stability, which the new teams will need to rebuild in their new form.

4.8.6.2 Shadowing

The shadowing strategy is based on creating 1 new team composed of the “new members”. During the ramp up phase, the team will closely *shadow* or follow an existing, experienced team in its duties in order to learn the product, the code, the rules, etc. The benefit of using this approach is the lack of need to break up, what could be a well performing, stable team. Naturally, the ramp up time will be much longer for the new team,

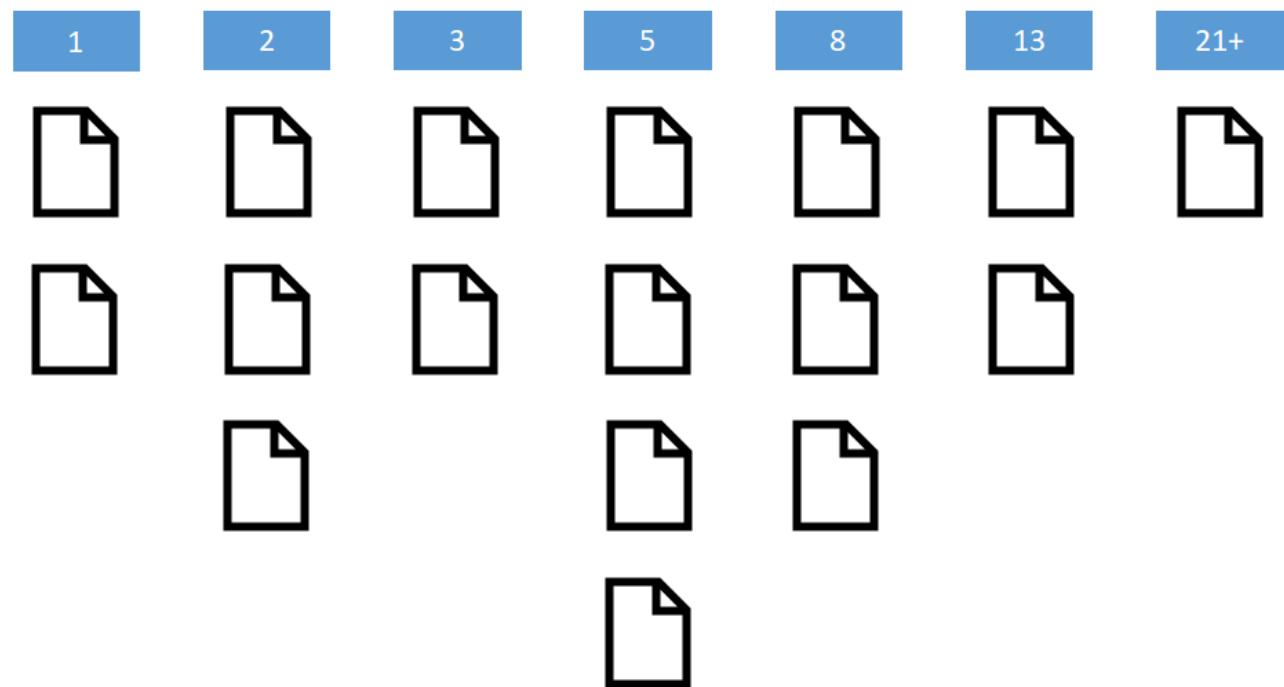
however this will occur without severe disruption (in reality some capacity of the existing team will be utilized by the new one) of the work currently being done by the existing one.

The choice of the proper strategy depends on multiple circumstances within the area, where the team changes are being conducted. An Agile Coach should be consulted should such need arise to evaluate the need for either one.

4.8.7 Team relative estimation synchronization

For those teams estimating their Product Backlog in story points, or another method of abstract, relative estimation, the most difficult part is getting a common understanding of what the given values represent across the team. Usually the team's perception of a common language of "story points" develops as the team continues to work forward, however it is quite helpful to conduct a synchronization session after the first several sprints, in order to speed up the alignment.

The input to the session is a set of user stories, which were estimated and completed by the Scrum Team in the last 6 sprints. Having done the user stories, the team is aware of the actual complexity, time and materialized risk that came with their delivery, hence has an experience-based reference in regard to their effort.



It's best to prepare for the exercise with print-outs of the user stories. Make sure to cover/hide the initial estimation that was done during planning (it can be revealed after the session to compare it with the post-session results). The synchronization session consists of 4 parts:

1. Taking the user stories and arranging them on a table or on the ground with respect to the **time** factor - depending on how time-consuming the user story was, from the left (least consuming) to the right (most consuming). After the stories were ordered mark each one with a number, starting from 1 (on the left), incrementing the value by 1 for each consecutive item to the right of the previous one. The label next to this number with "T" or "time" to identify the value with the variable.

2. Next, do the same with respect to the **complexity** factor, also assigning values as in the previous point, only this time adding the label with “C” or “complexity”.
3. Next proceed in the same way with the **risk** factor, assigning values and adding the label with “R” or “risk”.
4. Finally, take all the user stories, summarize the “points” and order the backlog items again on a table or on the floor with respect to the total number of points of all 3 factors, going from the left (least number of points) to the right (highest number of points). After ordering the items, create story point buckets with the following values: 1,2,3,5,8,13,21+. Assign the left most item to the (1) bucket and the rightmost item to the (21+) bucket. For the remaining items go one by one and discuss them together while attempting to continue the assignment of the remaining items in between the created buckets. The discussion here is crucial since it drives the main goal of the session, which is getting a common understanding of the reason behind the estimations by all team members.
5. In the end, uncover the initial estimates of the user stories and compare how you did as a team - just for reference, comparing the “before” and “after”.

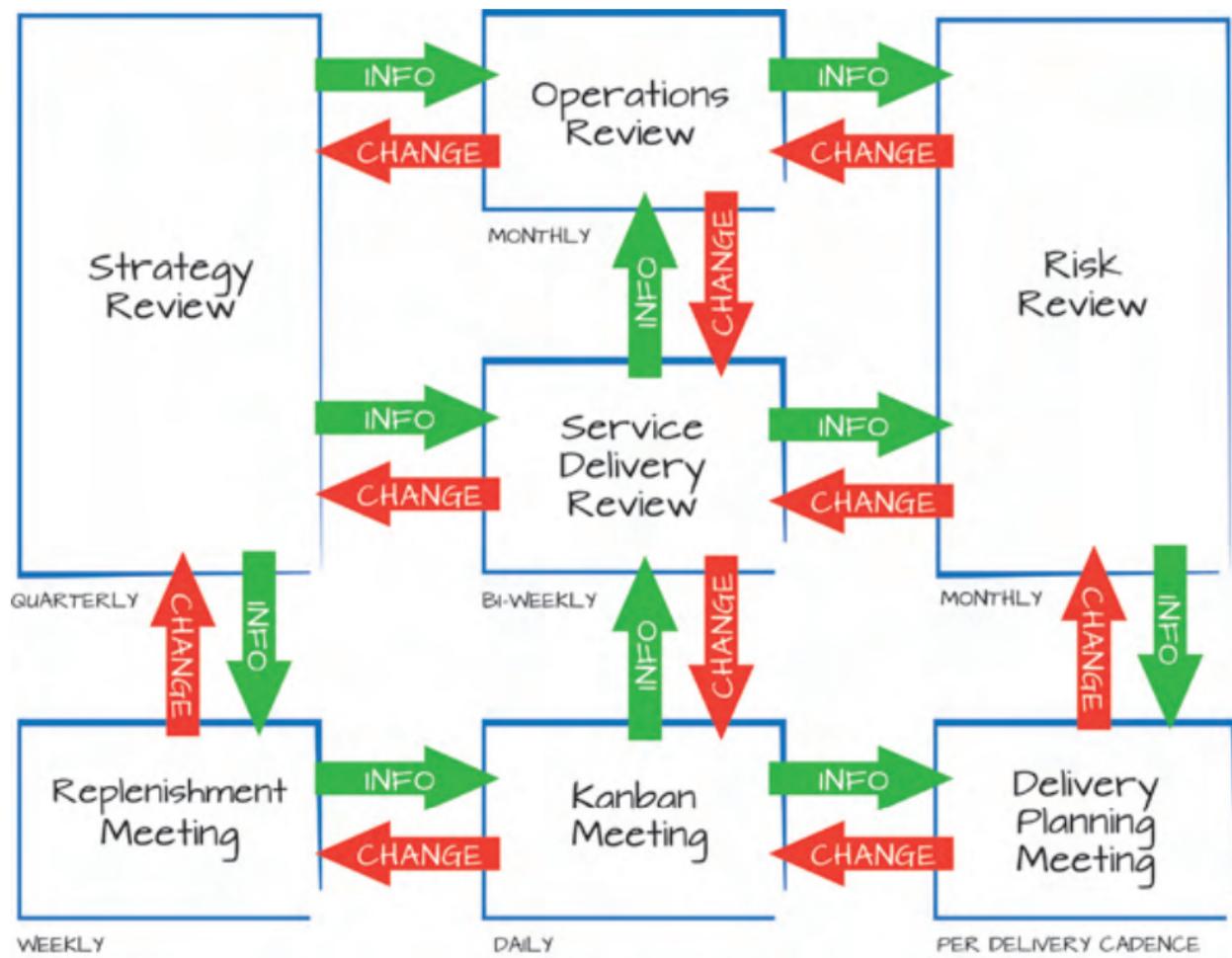
The user stories in the buckets (at most 3 from each bucket - if there are more, select the most representative ones) now constitute a set of reference user stories for the team! They may be used in the future by the team, during Backlog Refinement sessions and Sprint Planning to help gain a common estimation if it becomes a challenge for the team. The reference estimation set should be stored in a team space in Confluence and it is also good practice to have some print outs available for the team during the estimation sessions for support if necessary. As the team matures, the use of these reference user stories becomes less frequent or even unnecessary.

4.9 4.9 - Events for Kanban teams

4.9.1 Events in Kanban

All Events/ Meetings in Kanban are optional and should be decided by the team which should be used. However, they might be valuable from the "Implementing Feedback loops" perspective. Kanban defines seven specific feedback opportunities or cadences. Cadences are the cyclical meetings and reviews that drive evolutionary change and effective service delivery. “Cadence” may also refer to the time period between reviews—one workday or one month, for example: Choosing the right cadence is context-dependent and it is crucial to good outcomes. Too-frequent reviews may compel changing things before seeing the effect of previous changes, but if they are not frequent enough, poor performance may persist longer than necessary.

A scheme of seven cadences, depicted below, shows suggested frequencies for the reviews in a typical enterprise or multiple service context



(A set of cadences showing feedback loops infographic - <https://kanban.university/>)

The Kanban Meeting. Frequency: daily. Duration: 15 minutes

This is the (usually) daily coordination, self-organization, and planning review for those collaborating to deliver the service. It often uses a “stand-up” format to encourage a short, energetic meeting with the focus on completing work items and unblocking issues.

The meeting is similar to the stand-up meetings in Scrum. The Flow Master facilitates the meeting. Everyone in the team should participate because this 15-minute meeting allows the team to synchronize, observe and track the status of work items.

It observes the flow of work and takes into consideration decisions made at Replenishment and Commitment, Delivery Planning meetings, and Strategy Review. During this meeting, the team can present new information and check if they are within WIP limits. It also puts the focus on completing the tasks that are in progress before pulling new ones.

The goal of the meeting is to observe and track the status of work items and observe the flow of work. The team shares the progress to the Delivery Planning meeting and Service Review.

Replenishment Meeting. Frequency: weekly/as-needed. Duration: 20-30 minutes.

This meeting is for moving items over the commitment point (and into the system) and to oversee the preparation of options for future selection.

Kanban is a pull system that needs constant input in the queue, and a stocked and prioritized backlog. These actions are performed at the replenishment and commitment meeting. That is why this meeting should include the team, product owners and product development management. And anyone who can assess the technical or dependency risk and advice on scheduling, sequencing or grouping tasks.

This allows for the exchange of the latest project information and market data with everyone. The team shares relevant observations from the Daily Kanban and Service Delivery Review meetings. And also decisions and changes from the Strategy Review meeting.

The goal of this meeting is to make informed decisions, supported by the team and stakeholders. The decisions are shared at the Daily Kanban and enable prioritizing incoming work and keeping a steady flow of work moving through the Kanban board. And also ensures the team can deliver the committed tasks.

Delivery Planning Meeting. Frequency: variable – per delivery cadence. Duration: 1-2 hours

This is to monitor and plan deliveries to customers.

As the team moves tasks to the “Done” column, certain tasks go straight to delivery. Other tasks need to be handed off to other teams and departments. The Delivery Planning meeting reviews these finished work items and the tasks are due to be finished.

Product Owner facilitates the meeting. It includes the Flow master and everyone related to receiving and accepting the delivery. Managers, anyone involved in logistics, and specialist that have the technical knowledge and risk-assessment capabilities.

The Delivery meeting takes into consideration information from Daily Kanban meetings on which items are potentially ready. And any hand-off accounts and risk considerations about the items ready for delivery were discussed at the Risk Review.

The goal of this meeting is to ensure a smooth transfer of work in progress, and plan and decide which items to deliver. Decisions from this meeting are shared at Daily Kanban meetings. While issues are discussed at Risk Review.

Service Delivery Review. Frequency: bi-weekly. Duration: 30 minutes

This is to examine and improve the effectiveness of a service (this and subsequent cadences apply to a single service).

It doesn't matter how fast your workflow is, or how efficiently you work if the client is not satisfied with the result. The Service Delivery meeting is comparing customer expectations and the delivered product. It focuses on checking the team's performance against commitments, quality, lead-time, etc.

Again, the Product Owner conducts the meeting. It involves representatives of the team, customers, and other external stakeholders. The transparency and focus on the client's needs during this meeting are key for building trust with the client. During this meeting, the facilitator shares progress and data from Daily Kanban, decisions made at Operations Review, and actions from Risk Review meetings.

This meeting also looks at the team's capabilities, sets objective customer-focused metrics. Here, the customer and managers aim to balance demand, set reasonable delivery rates and evade unnecessary risks. The decisions and findings are then reported at Operations Review.

Operations Review. Frequency: monthly. Duration: 2 hours

This is to understand the balance between and across services, deploying resources to maximize the delivery of value-aligned with customers' expectations.

Since no team is an island, the Operations Review meeting takes a holistic overview of all internal teams and systems. The performance of the organization relies on optimizing each team and the organization as a whole system.

The Operations Review includes the Service Delivery and Flow master of each Kanban team, Senior Management, Senior Business Owner or customer representative. It also requires the presence of mid-level and functional managers. During the meeting, the different managers share relevant information. The findings from Service Delivery Reviews for all Kanban teams and systems. The business performance information from Strategy Review. And all ongoing initiatives from Risk Review that concern organization-level changes.

The goal of this meeting is to analyze the efficiency of different teams and the organization as a whole. Then, review the demand and capacity of each Kanban team, focusing on dependencies and their effects. It is also a good time to identify any underused capacity through the organization that can improve lead times. The improvement suggestions, decisions, changes and actions from this meeting are communicated at the Service Delivery Review and Strategy Review meetings. As well as at Risk Review meetings.

Risk Review. Frequency: monthly. Duration: 1-2 hours

This review is to understand and respond to the risks to effective delivery of services; for example, through blocker clustering.

Every work process is related to specific risks that influence delivery. In Kanban, the goal is to identify risks and bottlenecks before they substantially impact the workflow. Then, take steps to mitigate those risks.

The Risk Review looks at the issues identified at Operations Review and Service Delivery Review. As well as input from Delivery Planning meetings. Again, the Product Owner facilitates this meeting. In some instances, it can be a Kanban coach or Flow master. In the meeting should participate everyone who is familiar with the current recent blockers – team members and managers.

The goal of the meeting is to identify problems at all levels of the organization and identify the causes. Then, assess the risk associated with them and find a way to resolve and avoid the same problems in the future. The information from the Risk Review meeting is shared and discussed at Delivery Planning.

Strategy Review. Frequency: quarterly. Duration: 2 hours

This is for selection of the services to be provided and to define for this set of services the concept of "fit for purpose"; also for sensing how the external environment is changing in order to provide direction to the services.

The Strategy Review meeting is the highest-level meeting. It is concerned with reviewing and adjusting the overall business strategy based on the feedback from customers and the market changes. It also analyzes organizational capabilities and the main business goals.

A Strategy Review meeting takes the input from the Service Delivery Review and Operations Review meetings, combined with info from the Replenishment meeting. Participants of this meeting are senior executives, Product Owners and senior team members from customer-facing departments. The big-picture strategic goals and directions discussed at this meeting can be used for creating a Kanban roadmap.

The goal of the Strategy Review is to identify potential large-scale problems and find suitable solutions. Also, to course-correct team operations and optimize resource use where necessary. The information and decisions from Strategy Review are key for setting suitable KPIs, and holding successful Operations Review and Service Delivery Review meetings.

4.9.2 Scrum with Kanban

The flow-based perspective of Kanban can enhance and complement the Scrum framework and its implementation. Teams can add complementary Kanban practices whether they are just starting to use Scrum or have been using it all along. Kanban teams can also adopt scrum events for their needs.

Scrum Aspect	Short Description (According to the Scrum Guide)	How to apply to Kanban
Sprint	<p>Sprints are the heartbeat of Scrum, where ideas are turned into value. They are fixed length events of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint.</p> <p>All the work necessary to achieve the Product Goal, including Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective, happen within Sprints.</p>	<p>Kanban recommends most teams carry out planning/replenishment, delivery, and process retrospectives on a cadence.</p> <p>This cadence isn't mandatory and it is possible to carry out these activities "on demand." However, most teams simply do better on a cadence.</p> <p>The Sprint is a specific sort of cadence where the aim is to have a cross-functional team work together to complete the forecasted work. They focus on completing the work before taking on new work that will put the team's goal at risk—in essence, "cleaning the table" at the end of each Sprint. This encourages collaboration but can feel unnatural and wasteful to Kanban teams, especially if they already get the collaboration and swarming effects through their focus on flow and continuously working under a WIP limit.</p>

Scrum Aspect	Short Description (According to the Scrum Guide)	How to apply to Kanban
Sprint Planning	<p>Sprint Planning initiates the Sprint by laying out the work to be performed for the Sprint. This resulting plan is created by the collaborative work of the entire Scrum Team.</p> <p>Sprint Planning addresses the following topics:</p> <ul style="list-style-type: none"> Why is this Sprint valuable? What can be Done this Sprint? How will the chosen work get done? 	<p>Kanban teams that have already established an effective flow of work should carefully consider how Scrum Sprint Planning might be of benefit.</p> <p>Typically, Kanban teams don't invest much in estimating, preferring to break work down just in time. Therefore these teams would focus on Why the Sprint is valuable and What can be Done, and evolve the "How" throughout the Sprint.</p> <p>One of the key benefits of Sprint Planning is that it identifies a reasonable amount of work thus avoiding spreading ourselves too thinly. In other words, it is a form of limiting WIP. Good Kanban teams limit WIP already albeit in a different way.</p> <p>The other benefit a team gets from holding a Sprint Planning event is to come together as a team to craft a Sprint Goal. See below.</p>
Daily Scrum	<p>The purpose of the Daily Scrum is to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary, adjusting the upcoming planned work.</p> <p>The Daily Scrum is a 15-minute event for the Developers of the Scrum Team. To reduce complexity, it is held at the same time and place every working day of the Sprint. If the Product Owner or Scrum Master are actively working on items in the Sprint Backlog, they participate as Developers.</p>	<p>Good Kanban teams have a daily planning meeting in front of the Kanban board as their first-level feedback loop. Scrum and Kanban aren't that different in the high-level goal/purpose of this meeting.</p> <p>When it comes to running the meeting, there may be some differences.</p> <p>Kanban teams typically focus on the flow of work instead of the people doing the work. They work the board right to left focusing on flow problems.</p> <p>One Daily Scrum aspect that the Scrum Guide emphasizes is the focus on the Sprint Goal to make sure that tactical decisions are best aligned with the overall mission. Kanban teams would benefit from this higher-level focus beyond the immediate flow of specific work.</p>
Sprint Review	<p>The purpose of the Sprint Review is to inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work to key stakeholders and progress toward the Product Goal is discussed.</p>	<p>The Sprint Review is essentially an example of a feedback loop. Kanban teams could potentially just do this on-demand whenever some deliverable is ready for review. However, experience shows that having a cadence typically makes it easier to get the right stakeholders in the room and is overall more efficient and effective.</p>

Scrum Aspect	Short Description (According to the Scrum Guide)	How to apply to Kanban
Sprint Retrospective	<p>The purpose of the Sprint Retrospective is to plan ways to increase quality and effectiveness.</p> <p>The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done. Inspected elements often vary with the domain of work. Assumptions that led them astray are identified and their origins explored. The Scrum Team discusses what went well during the Sprint, what problems it encountered, and how those problems were (or were not) solved.</p>	<p>Most Kanban teams run retrospectives as well. Again, some teams do them on demand, but most teams would benefit from the discipline, simplicity, and predictability of having them on a cadence.</p>

References:

scrum.org

<https://kanban.university/#resources>

<https://kanbanzone.com/>

5 5 - Managing and using artifacts

- 5.1 - What is a product vs. a project (see page 150)
- 5.2 - Refining and estimating an initial Product Backlog (see page 152)
- 5.3 - The Product Backlog (see page 160)
- 5.4 - The Sprint Backlog (see page 171)
- 5.5 - The Definition of Ready (see page 176)
- 5.6 - The Definition of Done (see page 180)
- 5.7 - Using team metrics (see page 184)
 - 5.7.1 - Monte Carlo method. Statistical based forecasting for Scrum and Kanban (see page 193)
 - 5.7.2 - Configuring and using Kanban Team Metrics (see page 201)
- 5.8 - Team page (see page 211)

5.1 5.1 - What is a product vs. a project

In the contemporary IT world, the focus on delivery is shifting from a project view to a more product-related perspective. While for a long time the allocation of resources, budgeting and process management was maintained in project scope, an essential change came with the popularization of Scrum, KanBan and other Agile frameworks, promoting the product- and service-valued mindset.

The definition of Scrum states:

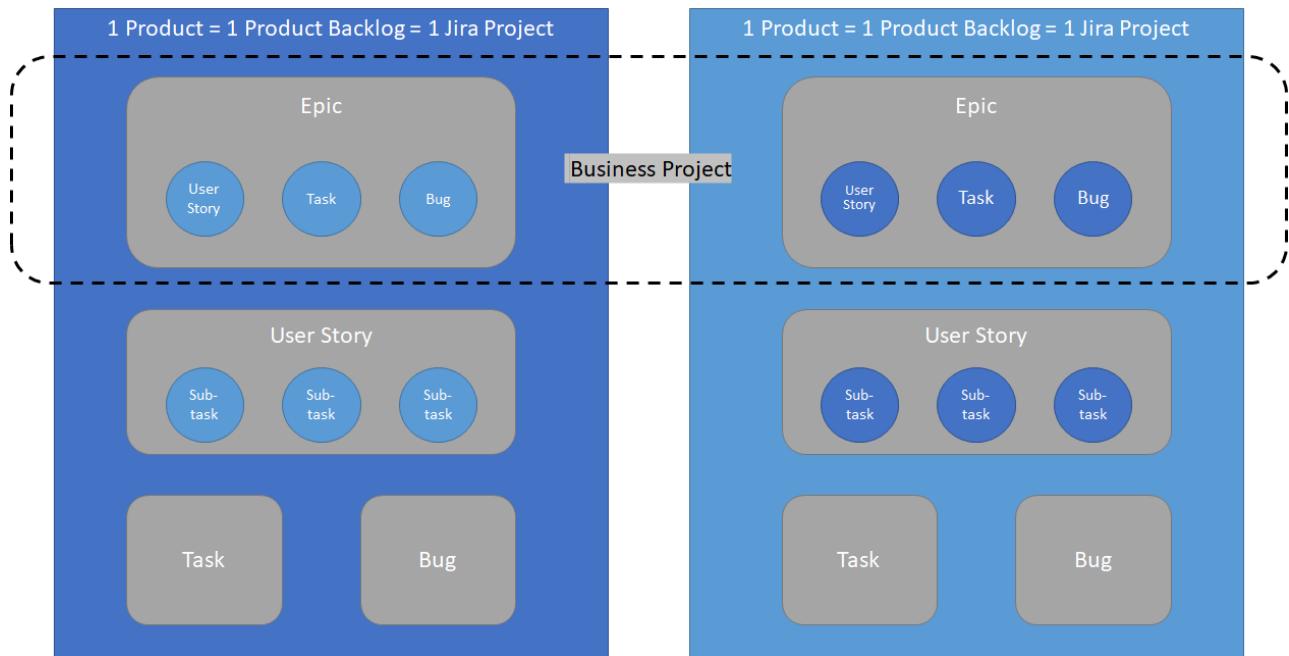
*Scrum (n): A framework within which people can address complex adaptive problems, while productively and creatively delivering **products** of the highest possible value.*

It is quite clear, that Scrum, including Small Scale Scrum places *the product*, as well as the value delivered with it, in the middle of all its rules and principles. Hence the teams and processes being built in EG Scrum are also having a product-centric approach and reasoning. Nevertheless, it is not that product is replacing the project, but rather the product becomes the first chair, while projects become natural entities to support their delivery with maximal value.

According to the KanBan University:

*... delays introduce risk to your ability to provide predictable and reliable **products and services**. The Kanban Method provides techniques to manage flow, remove delays, and get risks under control.*

While KanBan shares the product-centric approach around delivering high quality, it focuses on optimizing the flow of the process to ensure that the Service Delivery commitment is kept. This is done by balancing the demand and supply while keeping an eye on the end product, which KanBan redefines as a service to the customer.



5.1.1 The Product

A product in EG is an entity which:

- has internal or external users/customers
- delivers a measurable value to its users/customers and as well to EG as an organization
- has a defined name, purpose, a goal and a long-term vision
- has a dedicated Product Backlog in Jira, which specifies an ordered list of features planned to be delivered in scope of the product
- has a dedicated Scrum Team and a dedicated Product Manager in EG

In general, a product in EG is usually a system (simple or more complex, composed of components) or a piece of software that is being continuously developed, improved, maintained and delivered to one or multiple customers. Agile Teams in EG are being built around these products to allow flawless development and delivery with respect to Agile values and principles, as well as the highest focus on quality and value of the end product or provided service.

5.1.2 The (Business) Project

A business project in EG is a grouping of multiple Product Backlog items (more in: [5.3 - The Product Backlog \(see page 160\)](#)) (usually Epics, but may also include User Stories) across 1 or more *products*. The grouping is done with a Jira issue type *business project* not to confuse with the naming convention of a *Jira project* which is a technical representation of an EG product on the Jira platform.

The idea behind a *business project* is 2-way:

- to allow tracking of projects or initiatives within a product, which hold a separate budget and a separate group of stakeholders

- to allow management of projects or initiatives, which introduces changes across multiple products in EG

A business project may be defined in the scope of a single product, thus grouping a number of backlog items, such as: epics, user stories or bugs in order to reflect coverage of the scope against a defined budget and schedule. A business project might as well be defined across several products, when (for example) introducing an EG company-wide initiative, such as mandatory legal adaptations (i.e. GDPR) or parts of the EG company technological vision. This sort of grouping allows for better control of subsets of backlog items that share a common interest, as defined by the stakeholder.

5.2 5.2 - Refining and estimating an initial Product Backlog

A crucial part of working with Scrum or most Agile frameworks and methods for that matter is the management and maintenance of the Product Backlog (or another list of work items to be executed). Frequently, the following 3 words are used to describe the work, which is being done in reference to the backlog:

Triage - refers to the process of tidying the Product Backlog, including removal of irrelevant stories, prioritization, roadmap segmentation; in general anything that refers to the overall shape of the Product Backlog

Refinement - refers to the process of more precise specification and completion of the content of Product Backlog items, usually in their preparation for upcoming iterations & development

Estimation - refers to the process of evaluating the value of Product Backlog items, ideally with the use of *abstract, relative estimation* methods.

Maintaining the Product Backlog is extremely important for the sake of keeping a healthy product plan and long-term roadmap, hence, in turn, a healthy and valuable product. Conducting triage, refinement and estimation of the Product Backlog can be a bit specific in 2 different situations:

Product ramp-up - when a product is being started, there is a need to build an initial Product Backlog, triage, refine and estimate it, for Agile Teams to be able to start their work. It is sufficient to have only a part of the Product Backlog ready for the teams to start, but it is also valuable to have an overview of the complete, defined work to be known at the time being. Triage, refinement and estimation of an initial backlog are usually more high-level and not so detailed, due to the fact that the scope of the items, which need to be tackled is huge. It is impossible to accurately prepare the whole Product Backlog for the next iteration. That is why this stage should give everyone an overview and an introduction to what's to come.

Ongoing product development - during the normal operating mode, when Scrum Teams are working to deliver a potentially releasable product increment each Sprint or during upcoming delivery preparation by KanBan teams, the context of triage, refinement and estimation is a bit different. Firstly, the focus is usually mostly on the next upcoming Sprint or iteration (to have a list of items, which meet the Definition of Ready [5.5 - The Definition of Ready \(see page 176\)](#)). Secondly, the focus is on the next 1-2 Sprints or iterations after the next one, keeping in mind that some items need more time to be prepared than others ([4.2 - The Backlog Refinement \(see page 113\)](#)). While refinement of the Product Backlog can occur on a regular basis, either by the Product Owner him/her self (to the extent possible) and/or by the Developers, the estimation can only be done by the Developers without external interference. As compared to the product ramp-up phase, both refinement and estimation of the Product Backlog at this stage are much more detail-oriented, as the scope in question is also much smaller.

5.2.1 The approach in regards to the selected agile product delivery model

The notion of a Product Backlog, otherwise understood as an ordered list of work items that need to be completed for the product or service to provide expected value to the end-user, are very similar across most agile frameworks and approaches; hence the theme presented in this chapter can be treated universally. Both EG Scrum and Small Scale Scrum, as well as Consultancy teams who base their approach on these frameworks, have an identical understanding of the Product Backlog, as the list of all known items identified for the product to be completed in accordance with the existing expectations. KanBan's backlog is a bit more dynamic since the service-oriented nature of the KanBan method makes it less realistic to have a complete, defined scope of work to be known at any given time. Nevertheless, all known work to be done by a KanBan team, and each and every new learning that needs to land on the KanBan board, becomes a part of the living backlog artifact for a KanBan team.

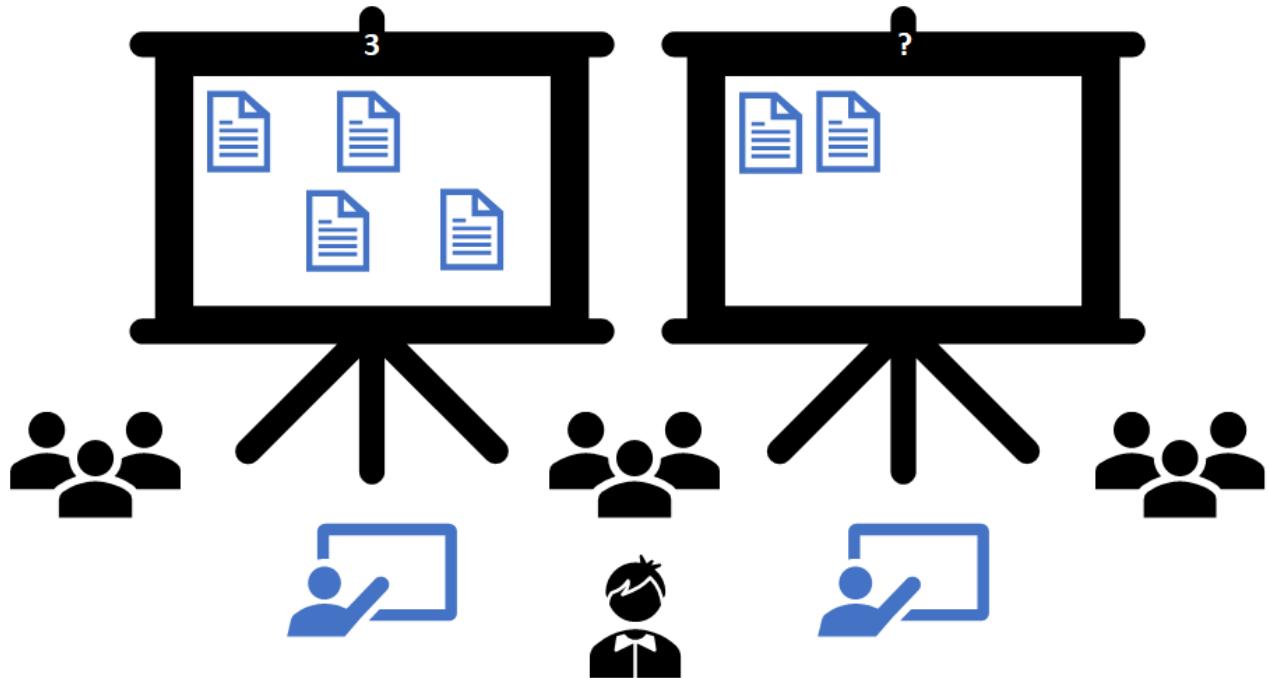
The moment when the Product Backlog becomes a bit more complex is in the case of High-Granularity-Teams, which work on delivering multiple products at the same time. Working on multiple smaller products is always a possibility for a single Scrum or KanBan team who can use their competencies to address the resolution of more than one complex problem. That being the case, in addition to refinement and estimation for a single product as mentioned in this chapter, the team(s) need to tackle the challenge of prioritizing multiple Product Backlogs to end up with a single ordered list for the team to work on. This is something that should involve the PM (preferably one) or PMs of the mentioned products to help identify the expected order in agreement with the forecasted roadmaps of the products.

5.2.2 Initial Product Backlog preparation

The journey to the first Product Backlog starts with the definition of the product to be built. Once the idea exists, there are multiple techniques for going from Product Vision to Product Backlog items (i.e. Vision-Box, more in [8.3 - Examples \(see page 427\)](#)). The fun starts with an initial Product Backlog that has been defined; at this point, the questions that come to mind include:

- *is this technically feasible to be done at all?*
- *how much of it can be done in time X? how much more in time X+Y?*
- *what dependencies exist regarding the scope?*
- *what competences are needed and in how many teams?*

In order to address most of these questions, it's a good idea to organize a full-day event, dedicated to triage/refinement/estimation of the initial Product Backlog items. The Agile Team(s) which will be working on the Product should participate entirely, along with the Product Owner(s), Scrum Master(s) and optionally an Agile Coach and a Product Manager.



5.2.2.1 Workshop agenda

1. Preparation: user story / epic print outs (the print out may be grouped into categories or epics if already broken-down to some extent), 8 pin boards (labeled with 1, 2, 3, 5, 8, 13, 21+, ?), pins, 2 white boards or flipcharts, markers; it is assumed that the teams have basic knowledge and have been trained in relative estimation previously to the event
2. At the beginning the Product Owner(s) / Product Manager introduces the idea and vision behind the Product.
3. First iteration: every Developer takes 1 user story print out from the first (category) pile, evaluates the user story and pins the card to the appropriate board defining its perceived estimation at the time being according to his/her best knowledge. The "?" board is for user stories which despite best efforts could not have been at all assessed
 - a. the Product Owner(s) is/are available in the room at all times to answer questions directly and individually
 - b. the evaluation can also be done in pairs to have a more precise result - this depends on the amount of scope to be groomed, the time that is available and the amount of participants; adjust accordingly
4. First iteration: After the first (category) pile has been depleted and all user stories are pinned to the boards there should be a short break and a time-boxed reflection. During the reflection, all participants are able to review the pinned user stories and re-pin them to another board if they feel the estimation should be different. Once the time-box has ended, the reflection stops. During the reflection, the Product Owner(s) are available to answer questions, now in a group manner.

5. Next iterations: the first iteration is repeated until all category piles have been depleted, following the same process: evaluation, break, reflection... During the next reflection phases, all user stories can be re-pinned, also those from previous iterations
6. During the whole exercise, major unresolved issues, questions, dependencies are noted on one of the available whiteboards or flip-charts
7. During the whole exercise, some core assumptions will need to be made and those major assumptions are to be noted on the second available whiteboard or flip-chart
8. At the end, the whole group dedicates more time to several user stories which have met at least one of the following conditions:
 - a. they are pinned to the "?" board
 - b. they have more than X pin marks (X needs to be defined after the exercise to describe a small percent of the user stories which have been re-pinned the most times)
9. The stories are discussed one by one together with the group to allow their clarification and initial estimation. If despite this additional round it is not possible, the user stories or epics are marked for the Product Owner(s) to revise or clarify offline, taking into account the gathered feedback.

What is achieved at the end:

- a rough estimation of a large scope of Product Backlog items, giving the best possible estimate at the time, which can help draw a roadmap on the basis of teams' velocities
- a more clear vision and better understanding of the product, the goal and the vision for the Scrum Teams
- dependencies/impediments associated with the Product Backlog items
- guidance/assumptions which need to be taken into account during further refinement of the Product Backlog

5.2.3 Defining the Minimum Viable Product within the Product Backlog

Agility focuses on incremental delivery; Scrum says: "*Inspect & Adapt*" - taking into account such fundamentals and applying the theory into practice, it is important not only to build working software at first but then to continue improving it. It's not about making everything perfect on the first attempt. Empiricism teaches us that we are supposed to evaluate what exists, what was created, and then build on it.

When planning the roadmap it is difficult to foresee all of the factors which may impact our product delivery, hence we estimate; we estimate what could be done, what needs to be done, and what must be done - taking into account what we know at the given point in time. Since the goal is usually to deliver as much as possible, but in a defined time period, it is vital to define an **MVP**. The **Minimum Viable Product** defines the scope of the Product Backlog which is mandatory for that Product to launch to production. The MVP is the smallest, most fundamental version of the Product, which has business value and can be made available to the customer. The MVP should be agreed with the Agile Team(s) starting work on the product, to give the Product Owner an assumption that this scope of work can be done in the defined time period with high probability. The MVP will be the first released version of the Product. Naturally, it is crucial to re-evaluate the feasibility of the MVP periodically, to identify any potential deviations; should that be the case, the scope of the MVP might need to be re-negotiated.



5.2.3.1 The MoSCoW technique

In order to establish the MVP, the Product Backlog needs to be prioritized. Having done that, having a roughly estimated Product Backlog and having the velocities of Agile Team(s) that will build the Product, it is possible to mark an approximate cut-off point which will define the MVP. But how to get the prioritization right? One of the methods to support backlog prioritization is called by the acronym **MoSCoW**.

Items in the Product Backlog should be evaluated with respect to 4 categories of the MoSCoW technique, to establish their relative priority:

Must have - defines an item, which is vital for the end solution (MVP) and must be completed in order for the end solution to be releasable and usable. This applies to all items, which i.e. are:

- core features of the product
- legal requirements
- security & safety requirements

Should have - defines an item, which has a high priority for the end solution and should be completed if possible (if time and resources allow). This applies to all items, which i.e. are:

- high value features of the product
- performance requirements
- efficiency and productivity related requirements (i.e. removing workarounds)

Could have - defines an item, which is considered to be valuable for the end solution, but is not necessary. Such as:

- desirable, "nice to have" features of the product
- fancy, visual, UX requirements

Won't have (this time) - defines an item, which will not be implemented in a defined time period, version, release, but may be considered a candidate for next iterations.

5.2.4 MVP in KanBan

While the notion of a Minimum Viable Product is an inseparable concept used with EG Scrum product delivery, its application in the KanBan Method may not bring the same benefit. KanBan is all about throughput, managing the flow and while the aspect of delivering a product or service with a focus on quality is still on the agenda, there is no specifically defined target. One of the main differences between Scrum and KanBan, is the lack of sprint goals which drive the focus for an upcoming iteration in the KanBan method. That means while there can still be multiple benefits in using MVP in KanBan delivery, it may vary depending on the type of work that is actually being done. Maintenance teams for example may not be able to define a valid MVP for their product at all, since their focus is on the delivery of service and support, rather than product increments.

It is always important to identify what do we want to accomplish with a defined MVP and the approximate time frame we hope to deliver it in, to see if this is a realistic approach for our team and product.

5.2.5 Estimating

A fundamental part of working with the Product Backlog involves some kind of evaluation of the effort of those items which need to be done. Estimations are used because it is difficult to define the exact metric (time, cost) for doing something when planning ahead. The closer to doing something, the better the estimates and they become even more accurate with every step during their implementation; empirically we find out what in facts is needed to complete the item, which was forecasted before.

Estimation consists of 3 factors that conclude to the overall EFFORT:

- **Time** - how time-consuming the task is
- **Complexity** - how complicated the task is
- **Risk** - what can go wrong, the uncertainty, the dependencies

A common way to express effort estimation in Scrum or other Agile frameworks is *relative estimation*.

5.2.5.1 Relative estimation

Relative effort estimation signifies forecasting the amount of work that needs to be done for an item to be complete, as compared to other items that potentially have been already completed (less than item X, more than item Y). During the estimation, we take the 3 factors into account: time to do it, the complexity of doing it, and the risk that comes with doing it.



Example: how can we estimate the effort of walking up the stairs to the top of some building in the city?

1. First, define a baseline - select a small building or house and define it as 1 (1 nothing, not story point, not hour, not day, just 1 - this is our baseline)
2. Now that we have something to refer to, we can look at another building (twice as tall) and can assume that this will be estimated to 2
3. We can then select another building, which is much further from us and it seems twice as tall as the previous one so we estimate it to 4
4. We get closer to the last building, and now we know that it was twice as tall from far away but from a closer perspective we feel it is 4 times as tall, so we change the estimation to 8 now that we know more, being closer to the building

Two popular estimation methods (depending on the purpose) can be distinguished:

- *t-shirt sizes* - high level estimation, usually used for estimating epics: XS, S, M, L, XL
- *story points* - more precise, used for estimating mostly user stories, but can also be used for epics (higher number 20+): 1,2,3,5,8,13,20,40,60,100 (modified Fibonacci) where
 - anything above 20 must be broken down into smaller stories (its an epic)
 - 13-20 may be considered to be broken down, but can remain if not possible
 - 1-8 represent healthy estimations for stories

- (i)** Fibonacci is used because the bigger the item is, the less detailed it is and hence the bigger the risk of incorrect estimations - that is why small values are closer to each other because they allow more precise estimation based on the small and detailed scope; the bigger they get the difference grows making the estimation more ambiguous

5.2.5.2 Planning Poker

One of the techniques used to evaluate the approximate effort of a Product Backlog item using story points is *Planning Poker*. Planning Poker is mainly used during backlog refinements and Sprint Planning events to estimate user stories (Product Backlog items in general).

Planning Poker can be conducted by various means, such as:

- with the use of physical cards, representing story points from the Fibonacci (or modified Fibonacci) sequence
- with the use of a smartphone app, representing the cards in a virtual way
- using a web tool or integrated add-on
- using fingers or chat room, if all of the above fail

There are 2 approaches to counting Story Points across Sprints:

A. completed item = 100% SP burned down, incomplete item = 0% SP burned down

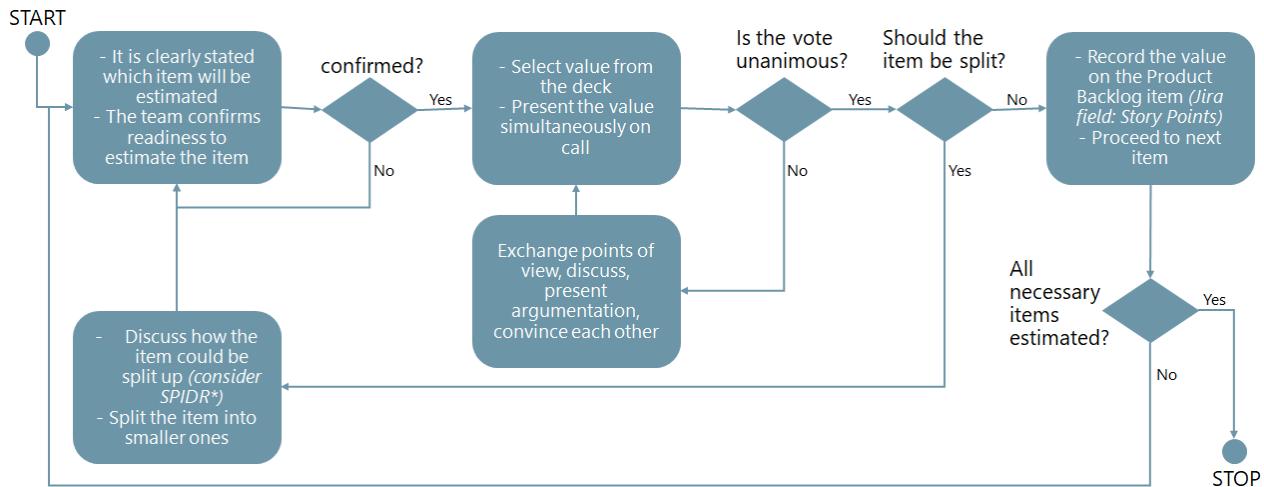
B. completed item = 100% SP burned down, incomplete item:

1. split into 2 parts: completed and incomplete, re-estimated separately then treated as in approach A

In EG, **approach A** will be used, as it brings the most simplicity and transparency to the value delivered in the scope of the Product Backlog. The amount of Story Points burned in a Sprint represents the **sprint velocity**, while the sum of SP burned during several consecutive sprints represents the **average sprint velocity** ([5.7 - Using team metrics \(see page 184\)](#)) – this is the one we want to focus on and stabilize in the long run (hence approach A suffices).

Running a planning session:

1. A typical set of Planning Poker cards consists of modified Fibonacci values and... a *coffee card*
 - a. The coffee card should be used by participants when the team needs a short break during the estimating session
2. The participants start by confirming which item will be estimated and confirm readiness to estimate the item
3. The participants select a value from the deck and raise the card at the same time (on call – for example 1... 2... 3!)
4. The participants mutually review estimates and discuss, justifying and attempting to convince each other until a consensus has been reached – repeat the voting if needed



5.2.5.3 Common mistakes

Some advice and a few common mistakes to avoid:

- during Planning Poker no other values should be used, than the ones defined for it
 - if you can't decide between 8 and 13, and you feel it should be an 11, choose the value closer to your estimate (in this case 13)
- the average value from all of the team members' estimates should not be calculated
 - the team needs to reach a unanimous consensus
- votes should not be conducted separately on different parts / technologies / components of a Product Backlog item
 - one item = common team estimate
- don't be afraid, don't worry, have courage to state your opinion
 - you can't be wrong - trust your "stomach" feeling

5.3 5.3 - The Product Backlog

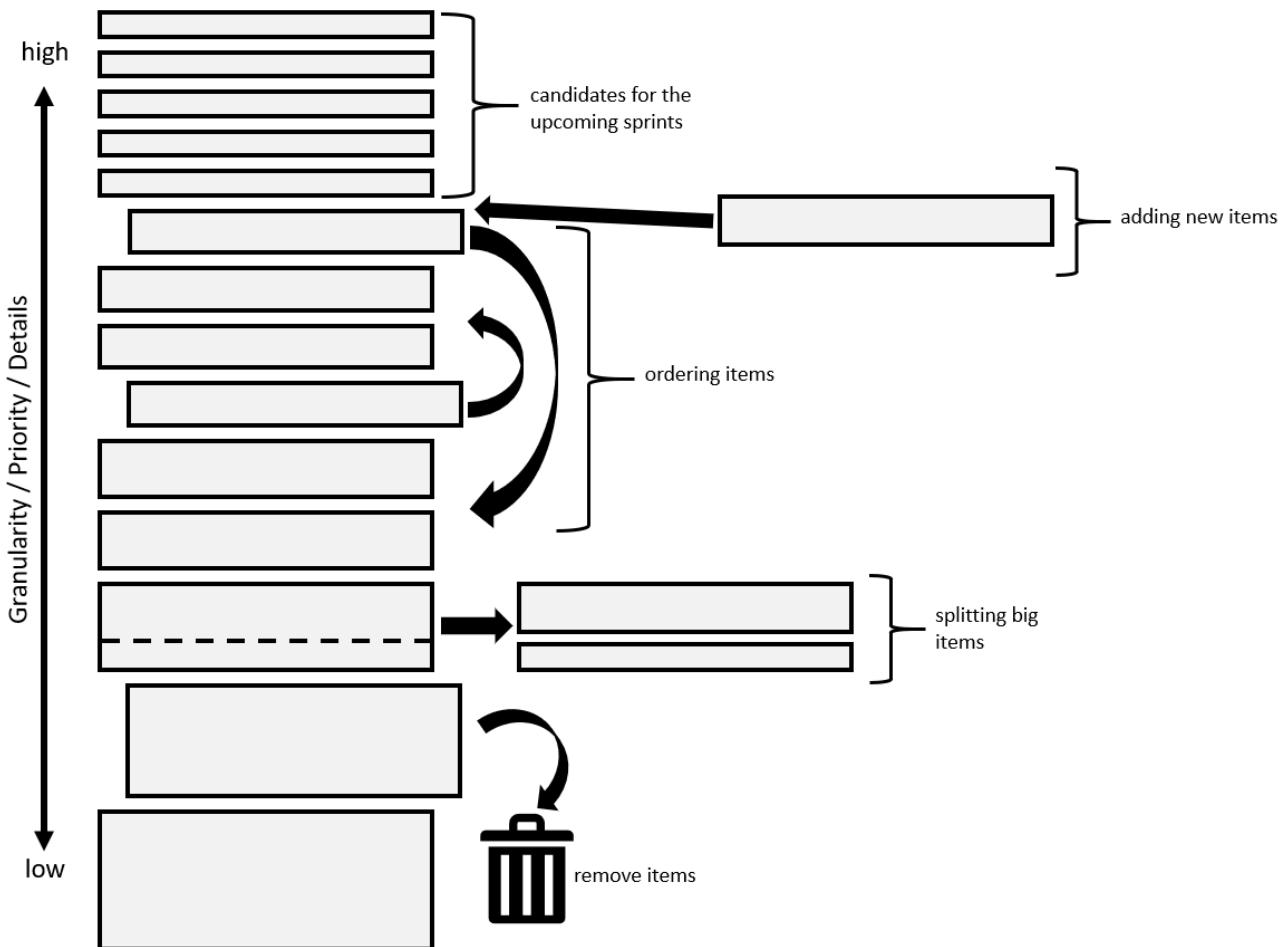
The Product Backlog is an ordered list of everything that is known to be needed in the product. It is the **single source of requirements** for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering. A Product Backlog is never complete. The Product Backlog evolves as the product and the environment in which it will be used evolves. The Product Backlog is dynamic. It constantly changes to identify what the product needs to be valuable, appropriate, competitive, and useful.

5.3.1 The Product Backlog overview

The Product Backlog is an ordered list. This order is being done in three dimensions: granularity, priority, and details.

- **Granularity** - during the process of preparing Product backlog items for sprints the top items are being sliced to smaller pieces to fit the Sprint (possible to be done within one sprint) and agreed within Scrum Team story size scale. On the contrary, the bottom items are big and most of the cases represent the idea or high-level requirements that can be in the future cut into smaller specific actions and changes during the process of preparation and moving the items higher in the Product Backlog.
- **Priority** - top of the Product Backlog contains items that are the most valuable and important features to be done by the Developers. They are the candidates for the next Sprint/Sprints. Most of the work in the Product Backlog is focused on top of it, so everything is prepared for the upcoming delivery. The priority order of the backlog is not fixed and may constantly change during the ordering items process. Product Backlog items can move both - up or down in the Backlog - so they reflect the current state of requirements. New items can be added at any time and the position in the Backlog informs everybody how important they are for Product development. The items, that we do not need anymore should be deleted so they do not rubbish the Backlog content.
- **Details** - the most specific and detailed items are situated at the top of the backlog - they have to be ready for Sprint execution. The bottom items are not yet planned for delivery and it is possible that they may not be delivered at all. At the given moment they are probably just specific ideas with a brief description and this may be enough for the Product Backlog.

A good, healthy well-ordered backlog should have specific, small detailed, and ready to go items at the top of it and bigger, generally described and idea-oriented items for the further future at the bottom. The backlog should be also up to date.



5.3.2 The Product Backlog management

The Product Owner is responsible for Product Backlog management which contains the following activities. Those activities may be done by the Product Owner himself or they can be the effect of Backlog refinement process conducted with Developers [5.2 - Refining and estimating an initial Product Backlog - Next Generation Agile - Confluence EG A/S \(see page 152\)](#):

- **Value management.** Since the Product Backlog represents all the upcoming work on the product it is essential that the Product Owner has considered which of the items have the biggest value to the customer. This would be a very important factor and input when executing the next activity.
- **Product Goal** describes a future state of the product which can serve as a target for the Scrum Team to plan against. The Product Goal is in the Product Backlog. The rest of the Product Backlog emerges to define "what" will fulfil the Product Goal. The Product Goal is the long-term objective for the Scrum Team. They must fulfill (or abandon) one objective before taking on the next.
- **Ordering the items.** Product Backlog is an ordered list. At the top of the Backlog, there are items that are the candidates for the upcoming sprints. At the bottom are User stories or even Epics that are manifesting some ideas and we need to work and collaborate under them to add clarifications and details to move them upper in the Backlog. So the order of the Backlog reflects our priorities and the

Product Owner should constantly review the Product Backlog Items and order them to comply with desired priorities.

- **Adding/removing items.** Product Backlog is a living artifact that should be constantly reviewed and updated. Usually, there is no problem regarding adding new items to it. Keeping the Product Backlog up to date means that the Product Owner should also clear it by removing the items which are no longer needed or we are sure that even if it would be nice to have the particular feature, we know that we will probably never have it. So don't forget to remove the obsolete items.
- **Items description and clarification.** It is essential to update the Product Backlog Items when we have anything new to be added to them. It is good to do it on a daily basis, because if we postpone we may miss something or we may not find enough time to focus entirely on Product Backlog Updates. The Scrum process also supports frequent Product Backlog update in the Refinement ceremonies. The focus point should be the top of the backlog because the top items are the candidates for the upcoming Sprint and should be as clear and ready for the Sprint as possible.
- **Decomposing bigger items into smaller ones.** This process usually happens when big stories or Epics are moving high enough in the Product Backlog that the Product Owner and the Developers are starting the refinement process. Due to fit big items into the Sprint time slot and to remove complexity of particular items, they need to be sliced into smaller parts. It is recommended to split the User Stories vertically (per functionality or devices being used) than horizontally per capability because each Story should deliver a working potentially releasable Product increment. Remember about the **INVEST** acronym which is described further on.

5.3.3 The Product Backlog Items

The Product Backlog contains Product Backlog Items. In EG we have *user stories* (sometimes gathered together in *epics* - very large user stories), *tasks*, *sub-tasks*, and *bugs*.

5.3.3.1 User Story

A user story is the unit of work that needs to be done. It represents the business requirement. Stories are the main components of Product Backlog. **User stories** are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

 As a < type of user >, I want < some goal > so that < some reason >

The type of user describes **who** is the final beneficent of Product improvement. This provides the Developers information about *who* (the personas) will use the described feature and whose problem we will solve with the User Story. A goal describes **what** needs, problems or set of brief requirements we want to cover from the type of user perspective by delivering the Story. A reason is a justification and the reason standing behind delivering of the User Story. It describes **why** we want to have our story to be delivered.

Below there are three examples of User Story:

- “As a HR Manager, I want to view a candidate’s status so that I can manage their application process throughout the recruiting phases.”

- “As a System Administrator, I want to have a possibility to search the users by name, Id number and status, so that I can quickly find the needed user.”
- “As a Legal Advisor, I would like to review the last published court decisions in the system so I can be up to date with the latest jurisdiction in the related legal field.”

The user story is a very brief description of the requirements. To make sure that the user stories are completed correctly and comply with a client's demands, each story contains also acceptance criteria.

Acceptance criteria are a formalized list of requirements that ensure that all user stories are completed and all scenarios are taken into account. Put simply, acceptance criteria specify conditions under which a user story is fulfilled. Concisely written criteria help Developers to avoid ambiguity about a client's demands and prevent miscommunication. Below you will

- **To define boundaries.** Acceptance criteria help Developers define the boundaries of a user story. In other words, acceptance criteria help you confirm when the application functions as desired, meaning that a user story is completed.
- **To reach consensus.** Having acceptance criteria synchronizes the Developers with the client. The team knows exactly what conditions should be met, just as the client knows what to expect from the system or application.
- **To serve as a basis for tests.** Acceptance criteria are a cornerstone of positive and negative testing aimed at checking if a system works as expected.
- **To allow for accurate planning and estimation.** Acceptance criteria scenarios allow for the correct division of user stories into tasks so user stories are correctly estimated and planned.

Below you will find an example of acceptance criteria from HR Manager story described above.

Ensure the HR Manager is able to:

- log in to the virtual job openings board system
- view / edit / add the status for job candidates
- update for each phase (e.g. Phone Screening Completed, In-person Interview Scheduled, Background Check in-progress)
- send email communication to staff regarding the process
- deleting candidates is excluded from the scope.

Although it is almost impossible to create a perfect, complete user story in one shot it is common practice to prepare the User Story incrementally, in three stages:

- The brief description of the story and sketch of acceptance criteria
- The conversations that happen during the ad hoc discussion, Backlog Refinement and Sprint planning to solidify the details of the story
- If needed, clarification and update/trade-off during the User Story development in Sprint.

5.3.3.2 How to write and prepare great user stories?

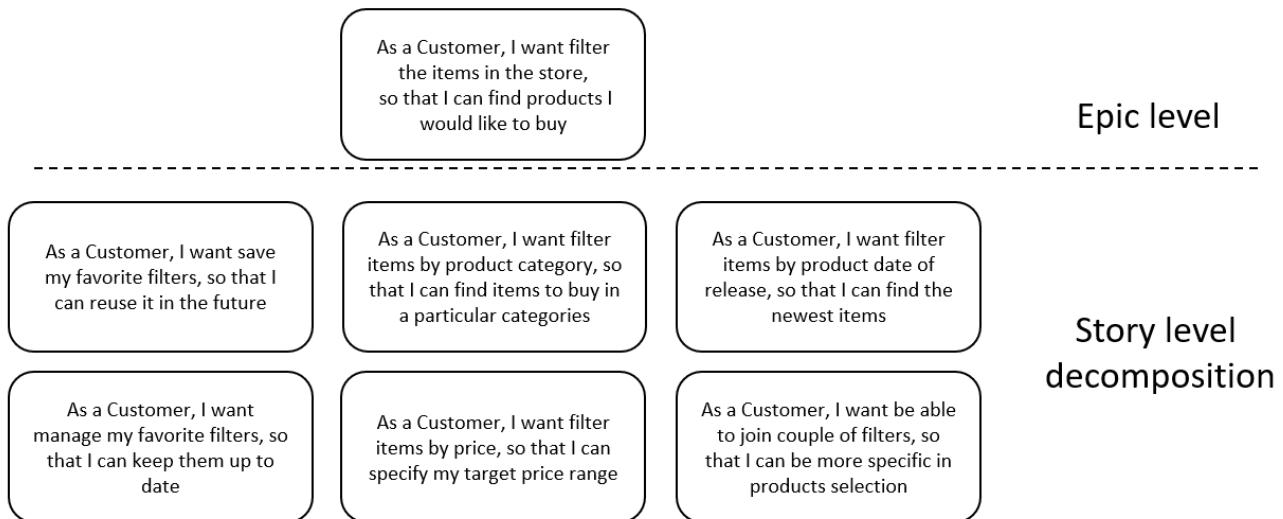
The **I-N-V-E-S-T** acronym helps to remember a widely accepted set of criteria, or checklist, to assess the quality of a user story. If the story fails to meet one of these criteria, the team may want to change it so finally, it is compliant with INVEST. All well-prepared user stories should meet INVEST criteria, which means that they should be:

- **Independent** - the user story can be fulfilled on its own and contains no dependencies to other items
- **Negotiable** - the scope of the user story focuses rather on *what* and *why* the Product Owner needs to achieve instead of how to do it (responsibility of the Developers), thus leaving room for discussion.
- **Valuable** - the user story must contribute to progress towards the product vision and must deliver value to the stakeholders
- **Estimable** - the user story must be described well enough to make it possible to estimate it
- **Small** - the work in the scope of the user story is decomposed and clear enough for the work to be done in one Sprint
- **Testable** - the user story goal and description must make it possible for the outcome to be tested by the Product Owner

5.3.3.3 Epic

An Epic is a body of work that can be broken down into specific small stories based on the needs/requests of customers or end-users. Epics are a helpful way to organize your work and to create a hierarchy (Epics are slots for multiple User-Stories). The idea is to break work down into shippable pieces so that large projects can actually get done and you can continue to ship value to your customers on a regular basis. Epic can be a feature, customer request or business requirement. Usually, Epics are too big pieces of work to fit one sprint time container.

Below you will find an example of Epic broken decomposed into smaller User stories:



5.3.3.4 Spike

Spikes are a type of user story used for complicated topics to do research and **analyze the next steps** that should be taken in future to deliver requirements. Spike's origins come from the Extreme Programming (**XP**) founder, Kent Beck, and rock climbing, as when you cannot climb due nothing to grab on to so you need to put a spike into the rock surface which forges a path for you to continue climbing on.

Spikes are used to gain the knowledge necessary to reduce the risk of a technical approach, better understand a requirement, or increase the reliability of a story estimate. Sometimes the Developers are unsure if they can complete the User Story or another item due to some potential blockers and probably can't even estimate it. Thus, you may consider a spike as an investment for a Product Owner to figure out what

needs to be built and how the team is going to build it. Spike is a great way to mitigate risks early and allows the team to ascertain feedback and develop an understanding of an upcoming Product Backlog Items complexity.

There are different uncertainties::

1. Why do we need it?

It should be always clear to the Product Owner but sometimes we need to analyze more deeply.

2. What do we need?

If we know the goal then we should find solution, e.g. using existing tools in ma

3. How to build it?

If we know the purpose / solution we might still need to analyze ways to develop / make it happen.

Example of cascade work in spike:

SPIKE - Research the most TOP3 accurate tracking tools		
	SPIKE - Check TOP3 tracking tools API if it's possible to integrate	
		Integrate chosen tracking tool with our tool

Specifics for spikes:

- they touch unknown area to make it more clear, we should if there is a lot of **uncertainty**,
- team may still deliver solution with spike if quick one or **workaround** was identified also,
- results of spike should be **documented** and may lead to implementation or another spike,
- **may be estimated** but should be at least time boxed to avoid never ending analyzes,
- acceptance criteria might be challenging (outcomes from R&D might be not reached yet)

Antipatterns using Spikes:

- Plan spike to answer “how” to build a solution without clear purpose “why”

- Plan spike without estimation or timeboxing and no effects as it's "still ongoing"
- Lack of demonstrable documentation from spike is a risk of knowledge lost
- Lack of clear requirements from business to be figured out by the team

Examples of spike:

Title: [SPIKE - Choose the most accurate from 3 tracking tools]

"In order to choose the most accurate tracking tool for runners, we want to check 3 different providers' trial accounts on our smartwatch during running."

Estimation - 3 SP

Acceptance criteria:

- Check solution no. 1
- Check solution no. 2
- Check solution no. 3
- Compare results

Title: [Enabler - Architectural design]

"In order to start work, developers need technical designs from an architect to know which stack to use and how".

Estimation - 5 SP

Acceptance criteria:

- Design diagram
- Documentation

5.3.3.5 Sub-task

A sub-task can be created for an issue (User story, Task or Bug) to either split the issue into smaller pieces or to allow various aspects of an issue to be assigned to different people. Subtasks are the particular pieces of work that are needed to be done, to complete the User story, Task or Bug. All Sub-tasks are entities that belong to their parent, under which they were created. Below you will find the characteristics of the Sub-tasks:

- A particular sub-task should be a piece of work possible to be done by a single person in a single day.
- All Sub-tasks are an essential part of their parent issue.
- All Sub-tasks are visible on the main screen of the parent issue.
- Sub-tasks always belong to the same project as their parent issue.
- Sub-task has all fields that are present in the standard issue.
- Sub-tasks cannot have a subtask of their own.

- Sub-tasks can be separately addressed and delivered during the delivery process.
- A parent issue can not be completed before all the Sub-tasks are finished.

Sub-tasks should be created during the second part of Sprint Planning to address the particular pieces of work needed to complete the parent issue (Story, Task or Bug). The responsibility for creating sub-tasks falls onto the Developers, which should identify all known tasks that exist at the time. While the team should aim to identify most of the work up-front, it is not possible to foresee everything. If additional pieces of work are discovered during the sprint, sub-tasks may be created and added to the backlog item, however, it should be verified prior if it does not affect the overall Sprint scope and Sprint Goal. An indication of a good break-down and earlier Backlog refinement sessions is a resulting item decomposition that results in 80-90% of sub-tasks being identified during Sprint Planning.

5.3.3.6 Task

A Task represents a set of engineering, investigation, proof of concept or technical debt work that is not directly related to a user story and is not visible as a product increment to an end-user. They should be treated as a separate item because they consume time and need to be planned, refined, estimated and tracked. Below you will find a specific sort of work, which can be addressed in the Tasks:

Training - if the particular team members need to share the knowledge and they are positive that it will take a substantial part of the Sprint time, a slot for this time allocation can be created under the Task. This can provide the visibility of such activities and time estimated to achieve it. Each such task should have a description of what knowledge will be shared and the desired output of the education process.

Retrospective improvements - Tasks are also being used for assigning and tracking the progress of the selected improvements, which are the outcome of the previous Sprint Retrospective meeting. This can provide the visibility and certainty that the improvement process towards Scrum perfection is addressed for the Team. A good practice is to assign the particular Team member to the Task so we have an advocate of the change, who will take care of the selected improvements.

The distinction between tasks and sub-tasks is that sub-tasks are decomposed pieces of work for particular backlog items and can't be independent items. On the contrary Tasks are independent items and pieces of work to be done. For better planning and execution during Sprint, Tasks can be split into Sub-tasks.

5.3.3.7 Bugs

Bugs or defects are items that represent an improper functioning, malfunctioning or lack of functioning of a feature expressed within a previously implemented backlog item. Handling bugs is a part of software delivery. We can distinguish several types of bugs that determine how we handle them in Scrum.

The first layer is to determine where the bugs were found:

- they were found “during the current Sprint” delivery and they are **related to** the items assigned to the **current Sprint**? or
- they are **not related to the current sprint delivery**.

The second layer of bug classification is answering the question of affecting the Product functionality:

- is the particular piece of Product **not working as designed**? or
- is it **not designed as it should work**?

Minding the above we can classify four types of bugs:

not working as designed	<ul style="list-style-type: none"> • fix as part of ongoing work • raise issue in Daily Scrum • discuss priority of the defect if big 	<ul style="list-style-type: none"> • create a Jira defect • PO determines priority • If relevant can be fixed in sprint; if needed discuss the trade-off's with PO
not designed as it should work	<ul style="list-style-type: none"> • communicate with PO immediately to discuss if we need to apply changes to the User Story • collaborate to adjust Sprint scope if necessary 	<ul style="list-style-type: none"> • create a Product Backlog Item • PO determines priority for future sprints • collaborate on the solution if needed

related to sprint item

related to product

Bugs related to a current sprint item that is not working as designed: When found during the review or testing they should be addressed in the current sprint to complete the story and meet its acceptance criteria. It is good to inform the Team during Daily Scrum and if the defect is big and can affect Sprint Goal then it should be discussed with Product Owner and the Developers.

Bugs related to a current sprint item that was not designed as it should work: This situation occurs when we missed some aspect of increment when designing the solution or we made other mistakes in design. We met all acceptance criteria but we know that the solution is not correct or complete. In such situations when a defect of this nature has been noticed, we should communicate the observation as soon as possible to the Product Owner and the team. The Product Owner and Developers may adjust the scope of the Product Backlog Item, make the design change part of a new Product Backlog Item, or simply leave it alone and wait for feedback at the Sprint Review.

Bugs related to the product which is not working as designed: A defect like this should be added to the product backlog, so the Product Owner can determine its importance and priority. If the defect is very urgent and important Product Owner can discuss with the Developers adding the defect into the current sprint. If needed, the Product Owner can discuss with the Developers what to descope in exchange to meet Team capacity.

Bugs related to the product which was not designed as it should work. We did the feature as designed but we didn't meet the customer expectations. The other example is that we did it, but other parts of the Product were negatively affected. When such defect has been found it should be added to the Product Backlog, prioritized by Product Owner and addressed accordingly.

It is essential to add a resolution type when solving a bug. For the available types please refer to [7.1.6 - Jira issue types and hierarchy \(see page 272\)](#).

Who should fix the bug? In Scrum, the Developers are responsible for the creation of working, potentially shippable product increment. But when handling bugs the rule is that if possible they should come to the creator of the faulty code if possible to be tracked. Such an approach supports taking responsibility for your own work and efficiency because creators are familiar with their own code and can fix bugs fast.

ServiceNow incoming Jira issues

A part of the Product Backlog can be build up by issues originating from 3rd party customer support systems, such as ServiceNow. In the end, such issues, whether classified as bugs, tasks or user stories become an integral part of the Product Backlog with ServiceNow being only the source system of the new bug or new feature. What characterizes such issues is usually an increased need for their refinement prior to considering them part of the main Product Backlog. These issues do not originate directly from the Product Owner or Scrum team, hence they may require additional assessment and communication with the support consultants to clarify their description or justification. All of the preparatory work done by the Developers on issues coming from external support systems should be included as part of the backlog refinement capacity that the team dedicates to preparation of the Product Backlog for the next Sprint Planning events.

5.3.4 Managing Product Backlog - tips for Product Owner

1. **Work with Product Backlog on a daily basis** – make it a habit. Product Backlog should be a living organism. This means that it should reflect the current state of the product requirements. Don't postpone the Product Backlog management activities so they cumulate and create a snowball. It will cost you a lot of energy to bring it back to life and you can always forget about something important. The Product Backlog is a transparent artifact and no one wants to review outdated information. It is hard to discuss the things that should be done when you cannot find them in the Backlog or the input is not relevant. Keep the backlog up to date so you are ready for discussion anytime.
2. **Let the others do Backlog management activities** - Since you are accountable you are also allowed to let the Developers do some of the activities i.e.: clarifications, updates, dependency identification, comments. This may prevent you to be a burden in backlog work engagement and will help the team-building process. You must also remember not to give anybody outside the Scrum Team credentials to manage the backlog. You can agree on the changes, but it is your final decision and job to update the Backlog.
3. **Focus on "what" and "why". Leave the "how" to the Developers** - You should be clear about what you want to be changed in the Product. Remember about the MVP concept [5.2 - Grooming and estimating an initial Product Backlog¹⁸](#) and the reason standing behind that. Don't forget to be specific about acceptance criteria, be aware of corner cases or alternative flows, express exclusions to be clear about the Jira item scope. On the other hand, try to avoid any suggestions on how you would like to have the change to be done. Leave it to the Developers – they know best how to do it in the most efficient way.
4. **Don't forget to delete obsolete items** - Many Product Owners remember about adding new features and Product changes but they tend to forget about deleting the items that became redundant. Don't keep the items that you know will never be done. Just get rid of it and keep your backlog clear and more manageable.

¹⁸ <https://egjira.atlassian.net/wiki/spaces/NG/pages/34603021/5.2+-+Grooming+and+estimating+an+initial+Product+Backlog>

5. **Keep the backlog visible, transparent and clear to the stakeholders** - When managing the Product Backlog Product Owner should keep in mind that the Product Backlog should be transparent and visible to the Stakeholders. This refers at least to clear Product Backlog Items description, up to date backlog order and content as well as clear priorities. This will greatly improve stakeholders and particular expectations management by having the whole picture in one place.
6. **Don't try to complete the product backlog scope** - The Product Backlog scope is never-ending activity lasting as long as the Product itself lasts. You should treat Product Backlog management as a constant activity instead of making it done approach. This can lead you to frustration and anger because the backlog will extend and keep changing all the time. Instead of a holistic approach, try to focus on close future activities and top backlog items, keeping in mind whole product vision.
7. **Remember it is all about maximizing the value** - The Product Owner is the value catalyst. He is speeding the value delivery reaction by ordering the backlog well. When doing such activities please remember not only about the revenue side. You must keep in mind also a cost side (this may come from the Developers feedback) and value the whole balance.
8. **Focus on collaboration** - The Product Owner is a person who is in the centre of all activities accumulating all the discussion with stakeholders and with the Team. Collaboration is a key factor for you to be successful. Discuss, collaborate, negotiate and learn to say "no" or "yes, but not now". It will help you a lot.

5.3.5 Kanban Backlog

Kanban backlog is represented as the "to do" column in Kanban board. The main difference is that there are no cadences/sprints. Flow is managed constantly, and backlog priorities should be also managed constantly. The above description (in regards to The Product Backlog overview, management, items), is applicable to Kanban teams.

5.4 5.4 - The Sprint Backlog

The Sprint Backlog is a set of Product Backlog items selected for the Sprint as a forecast for the scope of the potentially releasable Product Increment, which allows meeting the Sprint Goal. The Sprint Backlog is a plan by the Developers about what items will be delivered in the next Product Increment and the work needed to deliver that functionality into a "Done" status. During the Sprint, only the Developers can modify the Sprint Backlog; any changes must be discussed with the entire Scrum Team (with the Product Owner in particular), however, no changes can be done without the consent of the Developers.

5.4.1 Creating the Sprint Backlog

The Sprint Backlog is being created during the Sprint Planning Meeting where the Developers and the Product Owner are collaborating together to fulfil the Developers capacity for the upcoming Sprint. The candidates for the Sprint are the top Product Backlog Items. The items, which are being added to the sprint should be prepared during the Backlog Refinement process and should have "Ready" status [5.5 - The Backlog Refinement](#)

Definition of Ready¹⁹. If for some reason new items were introduced at the Sprint Planning first time, the Team should discuss the items and make them “Ready” to be able to add them into the sprint scope.

The agreed Spring Backlog consists of “Ready” Product Backlog Items (User Stories, Tasks and Bugs) and the plan for delivery. The active Sprint Backlog exists only for the duration of the Sprint. When the Sprint starts Product Backlog Items become Sprint Backlog Items. In particular situations - when the characteristics of the Team, the Product or number of defects inflow justifies such approach and it has been agreed in advance within the Scrum Team - during the Planning a small buffer can be left for the maintenance and defects handling.

To ensure continuous improvement the Sprint Backlog should include at least one high priority process improvement identified in the previous Retrospective meeting.

5.4.1.1 Multiple Scrum Teams delivering one Product

When multiple teams are delivering a single Product Increment, a Sprint Backlog is created per each group of Developers from the one common Product Backlog. One team should not have more than one Sprint Backlog per sprint to maintain focus and prevent priority issues. The items selected for the Sprint should not be dependant on other teams delivering the same Product. The proper preparation of the items, items selection and calibration between the Scrum Teams should be applied. For more details please read: [5.9 - Working with one product backlog across multiple teams](#)²⁰

5.4.1.2 One Scrum Team delivering multiple Products

In specific situations when one Scrum Team is working on multiple products, it is essential that the Team has only one Sprint Backlog despite being multiple Product Backlogs to maintain focus and prevent priority issues. **The amount of work related to different products should be limited to the maximum extent.** The priorities in such a case, as well as Sprint Goal, should be crystal clear to the Team to properly manage the Sprint execution.

5.4.2 Building a plan

When creating a delivery plan, which will allow the team to meet the Sprint Goal, some things should be considered:

- **how the particular Product Backlog Items should be delivered** - only the Developers know how to deliver a forecasted Product Increment during the Sprint. Usually, it is being done by decomposing User Stories, Tasks or even Defects into particular activities addressed in Sub-tasks or adding necessary comments or clarifications. During item decomposition, the Developers can dig in deeper into the details to have a better understanding of the work that needs to be done.
- **what items or activities should be done first to execute the Sprint in the most efficient manner** - sometimes it is better to deliver the Sprint Backlog in a particular order. For example, solving the Bug addressed within the Sprint can determine the faster delivery or different solution of another User Story or a Bug. In general, it is also good practice to consider priority and complexity - items that are high priority for the Product Owner, should be done first to maximize the chance to complete them;

¹⁹ <https://egjira.atlassian.net/wiki/spaces/NG/pages/35029022/5.5+-+The+Definition+of+Ready>

²⁰ <https://egjira.atlassian.net/wiki/spaces/NG/pages/34308149/5.9+-+Working+with+one+product+backlog+across+multiple+teams>

the same goes for complex items, which should be at the top of the Sprint Backlog to minimize the risk of failure due to the anticipated complex challenges.

- **can we achieve any synergies between particular Sprint Backlog Items** - during the Sprint Planning it can occur, that it is possible to re-use some of the solutions or do the particular work, that will benefit multiple Sprint Backlog Items. It is always beneficial to find all possible synergies and prepare smartly for the Sprint, so we are able to do more with less effort.

It is very difficult to foresee all the aspects of the work standing behind the User Stories, Tasks, and Bugs assigned to the Sprint. It means, that if new aspects of particular items occur (i.e. new work to be done or different solutions to be applied) the Developers or particular software engineers assigned should update the item and apply a new approach toward item completion. If needed the changes should be discussed within the Team, Product Owner or external parties. This is a live example of the Inspect and Adapt pillars in Scrum.

To improve crafting an accurate plan, for complex Sprint Backlog Items it is recommended to create Sub-tasks under the particular Sprint Backlog Items to address pieces of work needed to be completed. For more information please read: [5.3 - The Product Backlog²¹](#).

5.4.2.1 The Sprint Goal

The Sprint Goal is an objective set for the Sprint that can be met through the implementation of the Sprint Backlog. It is being crafted by the Product Owner after discussion with the Developers. It provides guidance for the Developers on why they are building the Product Increment. In other words - The Sprint Goal is a high-level summary of what the Product Owner would like to accomplish during a sprint, frequently elaborated through a specific set of product backlog items. A Sprint goal can help:

- Scrum Team deliver value to every Sprint
- Developers stay focused
- Product Owner determines the priority and manages stakeholders

A sprint goal is a short, one- or two-sentence, description of what the team plans to achieve during the sprint. It is created during the Sprint Planning meeting. The Sprint Goal gives the Developers some flexibility regarding the functionality implemented within the Sprint. As the Developers work, it keeps the Sprint Goal in mind. It is helpful when the changes or adjustments are needed to be done. Then if the Product Owner is not necessary or not available, the Developers know what direction to take when making decisions and applying the changes.

5.4.3 Managing the Sprint Backlog

Only the Developers are entitled and responsible for Sprint Backlog Management. It starts with the Sprint Scope agreed with the Product Owner. The Developers are responsible for crafting the plan for delivering the Sprint Scope (Product Increment) and making sure to meet the Sprint Goal. It is usually being done by decomposing User Stories, Tasks, and Bugs into smaller development activities addressed in Sub-Tasks. During Sprint execution the Developers are responsible for keeping the Sprint Backlog up to date which means:

- updating the current assignee of a Sprint Backlog item (also on Sub-Task level)
- updating and assigning the newly discovered activities needed to make the particular items Done
- discovering, discussing and addressing all issues and impediments related to the Sprint Backlog Items. It is especially done during Daily Scrum meetings

²¹ <https://egjira.atlassian.net/wiki/spaces/NG/pages/32276615/5.3+-+The+Product+Backlog#Sub-task>

- informing the Product Owner about all major risks in delivering particular Items
- discussing and agreeing on the Sprint Backlog Items trade-offs (adding, changing the scope or/and removing them from the Sprint Backlog) with Product Owner
- cooperating with all needed stakeholders or input providers (i.e. architects) regarding best possible Items delivery or issue solving
- making all needed decisions having Sprint Goal in mind when the Product Owner is not available
- meeting the acceptance criteria and the Definition of Done when completing the Sprint Backlog Items
- identifying and managing dependencies with other teams working on the same product.

5.4.4 The DOs and DON'Ts of a Sprint Backlog

- only the Developers can decide on updating or changing Sprint Backlog
- the Developers are responsible for keeping the Sprint Backlog up to date
- If a particular Sprint Backlog Item contains multiple technical work to be done, it should be broken down into Sub-tasks which lasts no longer than one day of work of a single Developer
- no significant changes to the Sprint Backlog Items (adding new items or removing the assigned ones) should be done without discussion within the Scrum Team (especially the Product Owner) and final agreement within it.
- The Developers should not remove any items from the Sprint even if they feel that they will not be "Done" during the Sprint duration, due to lack of time (overestimation). The Sprint should be completed (including all unfinished Sprint Backlog Items) when the time box assigned for a Sprint finishes. That gives transparency and input for improvement in the future.
- The Sprint duration can not be extended to make all the Sprint Backlog Items "Done" within the Sprint. The timebox for a Sprint is fixed and always lasts 2 weeks (by default in EG) or otherwise as agreed with the BU or team.
- During the Sprint, the Sprint Goal remains unchanged. If the Sprint Goal becomes obsolete, then that may be a basis for its cancellation.

5.4.5 Small Scale Scrum

All above is applicable in the Small Scale Scrum framework.

5.4.6 Kanban teams

There are no Sprints and Sprint Backlogs in Kanban. Sprint Backlog is scrum-specific. Kanban teams operate constantly (without mandatory cadences/sprints) by value delivering and flow management.

5.4.7 Consultancy teams

All the above applies to the Consultancy teams.

There is one aspect that might be challenging in the Consultancy teams setup and refers to the "customer-funded development" issue category. Some of the "customer-funded development" functionalities that we develop might be tested on the customer side (it depends on the agreement). If this is the case there might be some challenges regarding issues completion within the sprint (which we should aim for). It happens in situations where the customer's validation can't be completed by the Sprint ends.

The general rule here is that we should do as much as possible, to collaborate with customers in such a way, that all the job end-to-end is being done within the sprint. If this for justified reasons can't be achieved - there are two practices that can be applied:

- **Treat it as "done"** - In this approach, you are considering the issue "Done" as soon as all on our side is done and the issue is handled over for validation to the customer. Since this is the end of the development process on our side and we did the basic validation on our side, we can interpret that in our context this is the end of the process and the issue can be moved to "Done" status in Jira. This means as well that if any bugs are being raised by the customer after validation, will be created in Jira as independent issues linked to the main issue already moved to "Done".
- **Put it to a side** - If for any reason we don't want to apply the above rule in our context, there is an alternative that you can introduce. In the situation that we want to keep the issue open till the moment when it is considered done after the customer's validation, we can remain the issue open by the end of the Sprint. As soon as Sprint is closed the issues are being moved to the backlog or dedicated slot (sprint slot, i.e. "awaiting customer's validation"), waiting for customer's feedback. There are not being taken into the upcoming sprint, but the team is leaving some capacity free in the upcoming Sprint to have the possibility to add issues into the sprint if needed. As soon as the customer's validation results arrive and there is some work to be done, the team is taking the issue into the sprint and work under it for successful completion. This approach can be considered as a "Backlog management" activity that is being done by the Product Owner. This approach will reduce the negative impact of moving the "unfinished" issues from Sprint to Sprint and give you more precision in crafting the future Sprint's scope.

The above-mentioned solutions do not fulfill the whole spectrum of possibilities. If there is an alternative that can be successfully applied in your context - please consult it with a dedicated NGA Agile Coach and agree on the final shape to be implemented.

5.4.8 High Product Granularity teams

All above is applicable for the high Product Granularity teams however there are some areas that need to be emphasized:

- since in this category, we are having a high specialization level and not all the team members will be able to deliver the issues, we have to dedicate some extra energy to creating the plan for Sprint scope delivery. This means that we should take all the Sprint scope under consideration validating if we don't have any bottlenecks or big gaps between the team members handling the work between each other;
- when crafting the Sprint scope we should more closely look at the personal workload perspective to make sure that the scope is possible to be delivered;
- each Sprint is an opportunity to reduce the BUS factor, so to move forward we should do as much as possible towards knowledge share (i.e. by pair-programming);
- it is good to craft the sprint goal first and then build the scope around it. Otherwise, we might end up with too many distributed goals losing the focus point.

5.5 5.5 - The Definition of Ready

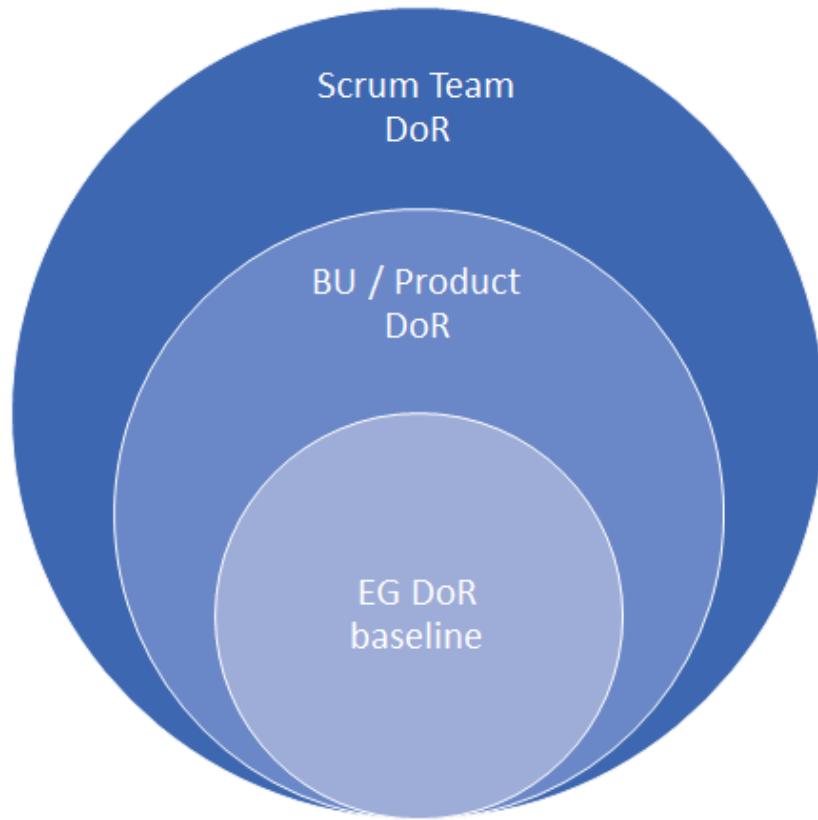
The Definition of “Ready” is the EG company-wide standard that describes the minimum requirements for any Product Backlog item to be considered *Ready* (see [7.1.7 - Jira issues workflow \(see page 279\)](#)) in order for work to be started on that particular item by a team. The DoR must be enforced and respected by the entire Scrum Team and surrounding stakeholders, as it is a critical element in the process for ensuring proper quality levels and reassuring that the work of the Developers will not be impaired during the Sprint.

The Definition of “Ready” is in close alignment to the 3 pillars of Scrum, where:

- **Transparency** - the DoR clearly shows what is expected and needed for work to be started on an evaluated Product Backlog item. This should be transparent to both the Developers doing the work as well as the Product Owner and the Stakeholders who request the work to be done. This definition should leave no doubt as to what is required and explicitly should show what pieces are missing for the expected work to begin.
- **Inspection** - can be interpreted 2-ways: on one hand, the DoR allows for verification of the work readiness by checking adherence of conditions for a Product Backlog item to be started. The entire Scrum Team should inspect if the DoR is met when refining the Product Backlog. On the other hand, the DoR as part of the process should be periodically inspected for flaws and potential improvements. This guarantees that the DoR is included in the continuous improvement as one of the process artefacts.
- **Adaptation** - the conclusion from the previous 2 pillars leads to improvement of the Product Backlog item’s state of readiness for beginning the work, as well as to adaptation of the DoR itself to better suit the process, quality and other defined metrics. Altering the prerequisites for any PBI is a part of that adaptation, as is the change of the DoR on both an organization and team level.

5.5.1 Levels of the Definition of Ready

The EG company Definition of “Ready” for Product Backlog items constitutes the baseline for all Scrum Teams to use when starting their work. The common DoR is defined on such level of detail, which allows compliance on an organization level independently of varying team/product standards. This is the starting point and absolute minimum of requirements to be used by all EG teams.



It is highly recommended, with growing team maturity and product awareness, to build on this DoR and expand it, making it more rigorous, more precise with respect to product specifics and even more compliant with team best practices, as well as standards. It is important to remember that the DoR should remain feasible to accomplish, meaning it should not be constructed in such a way to block ongoing and planned work.

When working with multiple teams on one Product Backlog it is mandatory that the same DoR is shared across all teams contributing to the Product. The understanding of the *Ready* state needs to be consistent across teams.

5.5.2 Working with the Definition of Ready

The Definition of “Ready” should be consulted mostly when refining the Product Backlog items (during Backlog Refinement sessions or during any refining work on the Product Backlog throughout the sprint), in order to check whether or not a particular item can be considered ready. Having met the DoR and estimated the Product Backlog item, it may be transitioned to the *Ready* status in the Jira Product Backlog.

This criteria can be once again ensured and verified during the Sprint Planning event, when the estimation and adherence to the DoR is confirmed before pulling an item into the Sprint Backlog. That way it is possible to identify any potential changes that might have been made to the prerequisites or to the item itself, before making a forecast for the upcoming sprint.

5.5.3 EG company DoR baseline

Due to the differing nature of Product Backlog items identified as Epics and those identified as User Stories or Bugs, the DoR baseline is divided into 3: the DoR for Epics, the DoR for User Stories and the DoR for Bugs, applicable respectively to the specific type of the evaluated Product Backlog item.

EG Epic DoR Baseline [Definition of Ready]

- The Epic is prioritized appropriately according to the Product Owner, with consideration of its risk, complexity and business value
- The Epic is aligned with the Product Manager's product roadmap and fulfills a part of the product's vision
- Initial, high-level constraints, risks and dependencies are identified for the Epic and listed in the Jira issue along with impact and proposed resolution
- The Epic complies with the high-level architecture and IT/Business solution in scope of the company product portfolio
- Acceptance criteria ([see 5.3 - The Product Backlog \(see page 160\)](#)) for the Epic are defined, listed and agreed upon
- High-level estimation (T-Shirt sizes or large Story Points) has been conducted and noted for the Epic (or in hours where abstract relative estimation is not used)
- At least 1 user story or workable Sprint Backlog item exists in the Epic in Jira

EG User Story DoR Baseline [Definition of Ready]

- The User Story is refined with respect to the **I-N-V-E-S-T** ([5.3 - The Product Backlog \(see page 160\)](#)) mnemonic for agile software development
- Detailed constraints and risks are identified and listed in the Jira issue along with impact and proposed resolution (including GDPR considerations)
- Sufficient acceptance criteria and/or description (including functional requirements) have been provided for the User Story to be deemed acceptable upon completion
- Should the User Story be accepted by a delegate of the Product Owner, then that person is explicitly indicated in the Jira issue
- If relevant for the User Story, performance criteria are defined along with the acceptance criteria
- If relevant for the User Story, visual / UX criteria are defined along with the acceptance criteria, such as UX visual designs, PSD/HTML files, etc. attached to the Jira issue
- If relevant for the User Story, integration specification and schema are defined along with the acceptance criteria, such as WSDL, XSD, XML files etc. attached to the Jira issue
- The User Story is estimated and sized in Story Points (or hours where Story Points are not used)
- The team is aware of how to potentially realize the User Story, sees no blocking impediments and has an idea how to present a live working demo of the User Story in scope of the next Product Increment

EG Bug DoR Baseline [Definition of Ready]

- The Bug is defined with a description of the flawed system behavior, indicating the steps, data and technical parameters necessary to reproduce the defect on a specified environment
- Detailed constraints and risks are identified and listed in the Jira issue along with impact and proposed resolution (including GDPR considerations)
- Sufficient acceptance criteria and/or description (including expected behavior or reference to the appropriate user story) have been provided for the Bug to be deemed acceptable upon completion
- Should the Bug be accepted by a delegate of the Product Owner, then that person is explicitly indicated in the Jira issue
- If relevant for the Bug, performance criteria are defined along with the acceptance criteria
- If relevant for the Bug, visual / UX criteria are defined along with the acceptance criteria, such as UX visual designs, PSD/HTML files, etc. attached to the Jira issue
- If relevant for the Bug, integration specification and schema are defined along with the acceptance criteria, such as WSDL, XSD, XML files etc. attached to the Jira issue
- The team is aware of the area affected by the Bug and has consideration for potential root cause and/or resolution of the defect; further investigation may be necessary to determine the issue during the sprint
- The Bug is estimated and sized in Story Points (or hours where Story Points are not used)

5.5.4 Small Scale Scrum

All above is applicable in the Small Scale Scrum framework.

5.5.5 Kanban teams

Definition of ready is a company-wide baseline for all teams including Kanban. From this perspective, the description above is applicable and requires individual interpretation taking to account the context of a specific team.

5.5.6 Consultancy teams

All above is applicable in the Consultancy teams.

5.5.7 High Product Granularity teams

All above is applicable in the High Product Granularity teams.

5.6 5.6 - The Definition of Done

The Definition of “Done” is the EG company-wide standard that describes the minimum requirements (functional, non-functional, visual, performance, documentation, etc.) for any Product Backlog item to be considered *Done* (see [7.1.7 - Jira issues workflow \(see page 279\)](#)). The Developers expresses the conformance of a Product Backlog item to the DoD by moving the item to *Ready for Review* Jira status, while the Product Owner confirms the conformance after verification with status *Done*. The DoD must be enforced and respected by the entire Scrum Team and surrounding stakeholders. It is a vital part of EG Scrum processes and a gateway for ensuring that a work item may be part of a *potentially releasable Product Increment*.

When a Product Backlog item or an Increment is described as “Done”, everyone must understand what “Done” means.

...

The purpose of each Sprint is to deliver Increments of potentially releasable functionality that adhere to the Scrum Team’s current definition of “Done”.

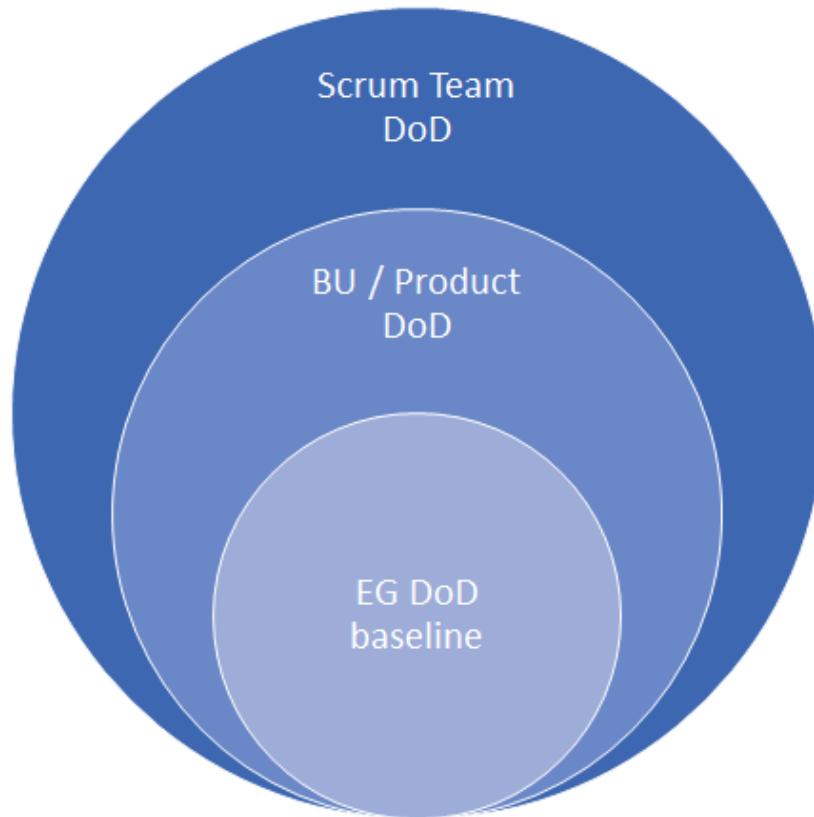
The Scrum Guide, Scrum.org

The Definition of “Done” is in close alignment to the 3 pillars of Scrum, where:

- **Transparency** - the DoD clearly shows what is expected for a Product Backlog item to be considered complete and to be part of a Product Increment, which might be released. The conditions for approving an item need to be coherent, unequivocal and agreed by all parties involved in the work in any way. This ensures that there is no misunderstanding when it comes to the final result, and the result is successful or not is perceived as such by all parties. It should be fully transparent what is missing in an item for it to be deemed as *Done*.
- **Inspection** - the DoD is the basis for one of the Scrum pillars, by creating conditions to perform verification of a Product Backlog item which has been assumed to be completed or is on its way to completion. The DoD statements should be checked frequently as guidance towards the Sprint Goal and ultimately before delivering the item with the potentially releasable Product Increment. Similarly to the DoR, the DoD can and should be inspected itself, especially with growing team maturity, to evolve the inspection process with respect to continuous improvement.
- **Adaptation** - the conclusion from the previous 2 pillars leads to improvement of the Product Backlog item’s criteria of completion, as well as to adoption of the DoD itself to better suit the process, quality and other defined metrics. Altering the *Done* criteria for any PBI is a part of that adaptation, as is the change of the DoD on both and organization and team level.

5.6.1 Levels of the Definition of Done

The EG company Definition of “Done” for Product Backlog items constitutes the baseline for all Scrum Teams to use when finishing their work. The common DoD is defined on such level of detail, which allows compliance on an organization level independently of varying team/product standards. This is the starting point and absolute minimum of requirements to be used by all EG teams.



It is highly recommended with growing team maturity and product awareness, to build on this DoD and expand it, making it more rigorous, more precise with respect to product specifics and even more compliant with team best practices, as well as standards. It is crucial to remember that the DoD should remain feasible to accomplish, meaning it should not be constructed in such a way which would hinder or block completion of work, unless it is strictly related to the expected criteria for releasing the Product Increment.

When working with multiple teams on one Product Backlog it is mandatory that the same DoD is shared across all teams contributing to the Product. The understanding of the *Done* state needs to be consistent across teams.

5.6.2 Working with the Definition of Done

The Definition of “Done” should be consulted most frequently during the following EG Scrum events:

- *Sprint Planning* - the DoD serves helpful artifact in supporting the team’s understanding of what exactly needs to be done regarding a Product Backlog items, so that they could properly estimate the effort required, identify the needed tasks for the “how” part and decide how much of the work they are able to do in the next sprint.
- *Daily Scrum* - the DoD serves as guidance toward the Sprint goal, as the team can and should inspect their work with respect to the DoD. By doing that they are able to identify remaining work and check their Sprint Backlog items for completion.
- *Sprint Review* - the DoD servers as a benchmark for verification of the completed Sprint Backlog items. Its understanding should be consistent for the entire team and bring transparency to the expectations, as well as state of implementation. If despite adhering to the DoD, the quality of the

product is underwhelming it might lead to a conclusion that tightening of the DoD is necessary and should be considered by the Scrum Team.

- *Sprint Retrospective* - during the Sprint Retrospective, the team can among other topics, discuss the evolve the Definition of “Done”. This could be either caused by relevant feedback from the stakeholders or by growing team maturity. In the later case, the team seeks to build on their DoD by tightening the criteria, making it more strict with positive impact on the outcome and overall product quality; that being of course part of continuous improvement. It is recommended however, to first reach a level of proficiency in using and applying the basic Definition of “Done” before proceeding further.

It's correct - the DoD lives throughout the entire Scrum process, allowing for planning, guidance and verification of Product Backlog items with respect to the Sprint Goals and the Product Vision.

5.6.3 EG company DoD baseline

Due to the differing nature of Product Backlog items identified as Epics and those identified as User Stories, the DoD baseline is divided into 2: the DoD for Epics and the DoD for User Stories, applicable respectively to the specific type of the evaluated Product Backlog item.

EG Epic DoD Baseline [Definition of Done]

- All underlying User Stories of the evaluated Epic have been Done, according to their Definition of “Done” including any type of testing, documentation, environmental dependencies and other deliverables as defined by it
- The complete set of Acceptance Criteria, as well as the goal of the Epic have been met and approved by the Product Manager and Product Owner
- Communication to the stakeholders with respect to the Epic has been ensured and completed

EG User Story DoD Baseline [Definition of Done]

- The entire solution code has been checked into the appropriate code repository defined for the product
- The User Story has been tested, including but not limited to: unit tests, integration tests, UI tests, UAT tests and regression tests, as defined for the product
- No blocking or critical defects remain open for the evaluated User Story
- All necessary documentation, as needed by the product or mentioned in the Acceptance Criteria, has been created and delivered to the location specified for the product
- All commits for the User Story have obtained a necessary amount of pull request approvals
- The complete set of Acceptance Criteria have been met and approved by the Product Owner or a defined delegate of his/her choosing

- The solution is deployed as part of a defined Product Increment or Version on a specified environment for the Product
- The GDPR considerations are specified and approved by the Product Owner or a defined delegate of his/her choosing.

EG Bug DoD Baseline [Definition of Done]

- The entire solution code has been checked into the appropriate code repository defined for the product
- The Bugfix has been tested, including but not limited to: unit tests, component tests, integration tests, UI tests, UAT tests and regression tests, as defined for the product
- All necessary documentation, as needed by the product or mentioned in the Acceptance Criteria, has been updated and delivered to the location specified for the product
- All commits for the Bugfix have obtained a necessary amount of pull request approvals
- The complete set of Acceptance Criteria (including the expected behavior) have been met and approved by the Product Owner or a defined delegate of his/her choosing
- The solution is deployed as part of a defined Product Increment or Version on a specified environment for the Product
- The GDPR considerations are specified and approved by the Product Owner or a defined delegate of his/her choosing

5.6.4 Small Scale Scrum

All above is applicable in the Small Scale Scrum framework.

5.6.5 Kanban teams

Definition of done is a company-wide baseline for all teams including Kanban. From this perspective, the description above is applicable and requires individual interpretation taking to account the context of a specific team. Additionally, for Kanban teams, it's a good practice to establish a "definition of pull". It describes criteria that need to be kept before pulling items between specific stages on the kanban board. Definition of pull should be established by the team and should be strengthened as the maturity of the team grows.

5.6.6 Consultancy teams

Il above is applicable in the Consultancy teams. For the "customer-funded development" issue category development, one remark can be applied to the issues that the testing is being made on the customer's side. The details have been described in the [5.4 - The Sprint Backlog \(see page 171\)](#) chapter.

5.6.7 High Product Granularity teams

All above is applicable in the High Product Granularity teams.

5.7 5.7 - Using team metrics

Many people assume that Agile and the Scrum framework ignores planning, using metrics, measuring KPIs. This, of course, is not true. Scrum only focuses on the fact that it is very difficult to plan complex products with a high level of detail up-front - it is better to focus on what we know and plan around that in a detailed manner and plan the rest on a high-level, just to plan in more detail when we know more about what's to be done.

There exist several metrics, which have proven to be very valuable to Scrum Teams in planning and improving - only when used properly. The purpose of team metrics isn't accountability, status management nor input for status reports. The purpose is to help the team plan better, to identify and address their weak points, to self-inspect and adapt. Naturally, information coming out of team metrics is a valuable source of information on:

- how the team is developing as a group?
- is the team on a good way to reaching stability?
- is the team's productivity growing or decreasing?
- what impact did any changes or external factors have on the team performance?

Based on that information, an Agile Coach or the team's manager can make decisions to support the team and help in reaching their goals, if they are facing unresolvable impediments.

Remembering the Agile Manifesto:

“...individuals and interactions over processes and tools...”

it is crucial to treat team metrics accordingly - as a means to help teams develop and improve, rather than monitor them and hold them accountable for the results.

5.7.1 Velocity

Velocity is most likely the single most relevant metric for a Scrum Team to use. Velocity, in general, represents the speed at which the team is moving forward.

Velocity...

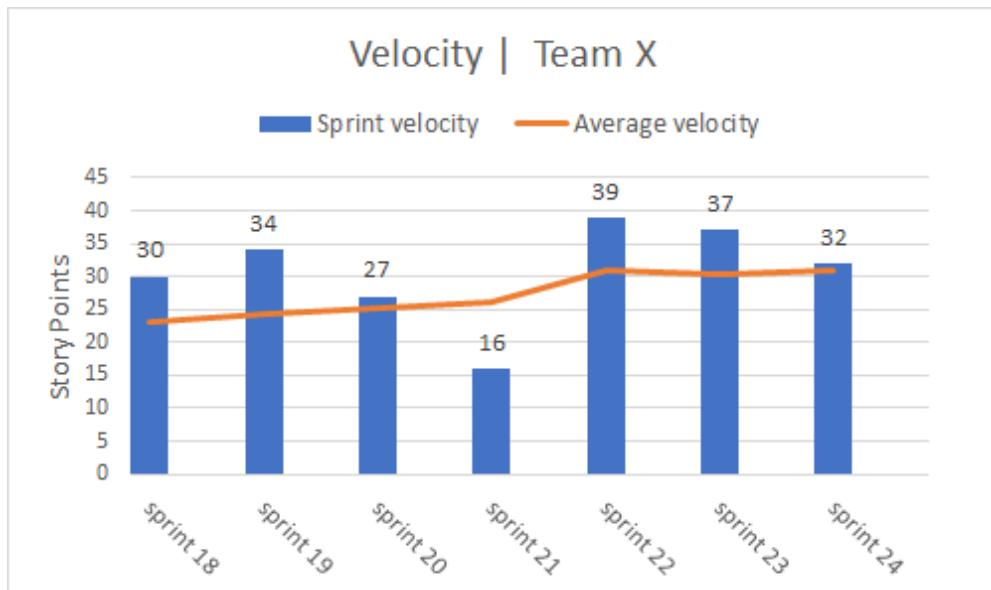
- is abstract and team-specific
- is not comparable between teams (to some extent)
- is most valuable to the Scrum Team

Tracking velocity helps in establishing mainly 2 things:

- a better plan, forecast of when a particular team might be able to complete an assessed part of the Product Backlog
- an indication that something is changing (good or bad) within the team or outside it, with direct impact on their performance

Often, the velocity is represented in Story Points (see: [5.2 - Refining and estimating an initial Product Backlog - Next Generation Agile - Confluence EG A/S \(see page 152\)](#)), which are being used by the team for estimating the Product Backlog Items. To be specific, a Scrum Team can focus on 2 types of velocity (4 in total taking into account the Capacity aspect - please see next part "Capacity"):

- **Sprint Velocity** - is an instantaneous value resulting from the number of Story Points, which were *burned down* (or completed) in the last sprint.
- **Average Velocity** - is an average out of a series of consecutive Sprints of the Scrum Team, usually limited to 6-12 last Sprints. The average velocity, as compared to Sprint Velocity, does not show the change of value from Sprint to Sprint, but rather a trend in which the Scrum Team is heading.



Both of the values provide some information on the team's wellbeing, however the average velocity informs us whether the team is stable or not - and that is a piece of key information. Average velocity is prone to minor deviations in the sprint to sprint velocities, so unnecessary panic is avoided once a "bad sprint" occurs once in a while. The average velocity also informs us whether any changes that occurred in or around the team, had a positive or negative impact on their performance.

In addition to being a sort of a health monitor for the team, the velocity statistics constitute an irreplaceable tool for supporting the team's estimation efforts. A team that reached its stable velocity or is near it (the amplitudes of the sprint velocities are minimal) may use their average velocity as a reference for their target in the upcoming sprints - as a facilitation mechanism.

A good practice for teams, that are more mature and have a stable velocity for a longer period of time, is to challenge it once in a while. Often teams stick to their velocity as it works, while their potential grows as they learn to work more effectively together. When things start being easily deliverable within sprints, it's worth raising the bar just by a bit and see how that works out. If it does - great! Try a bar the next time. If not - don't worry, go back to the previous assumption and try again in another while.

5.7.2 Capacity

The second valuable metric for a Scrum Team is its *capacity*. Capacity is simply the amount of work that can be handled by the team in the sprint. That amount of work is usually expressed as a ratio of hours available in the sprint-defined time period and hours that the Developers plan to dedicate for executing sprint-related tasks.

There are several differences between the capacity and the velocity:

- velocity is calculated post-sprint, while capacity is planned before the sprint (and optionally verified after if any changes occur during the sprint, in order to maintain historical accuracy)
- velocity is general for the whole sprint - Scrum events, internal events, and any other this which are not in the sprint backlog does not change the velocity, however, they do change the capacity

Some tips on how to calculate capacity:

1. Baseline hours - the baseline number of hours in a sprint is calculated by taking the overall, regular number of working hours x number of days in the sprint, and subtracting the constant, recurring events (sprint change) and hours which reduce the daily working time (see below - effective work time). In the end, that number is calculated by the number of Developers. For example:

```
10 (days) x 8 (hours) = 80 (hours)
80 (hours) - (10x1 hour) {daily effective time = 7, so 8-7=1} = 70 (hours)
70 (hours) - 6 (hours) {sprint change} = 64 (hours)
64 (hours) x 6 (people) = 384
```

Therefore, by having 6 (people) x 64 hours:

The baseline capacity **for** each 2 week sprint **for** a 6-Developers is 384 hours

2. Actually planned hours - the planned hours is simply the sum of the numbers of hours that all of the Developers plan to spend on work in the upcoming sprint. In order to calculate this for each team member, he/she must take the sprint baseline for 1 person and subtract the time planned on such events as:

- planned holidays / leaves
- planned workshops / trainings / conferences
- planned major travel that influences the regular work time

Such information should be provided by each Developer to calculate the overall planned hours for the team and in the end the team's capacity for sprint X. For example:

Following the previous example:

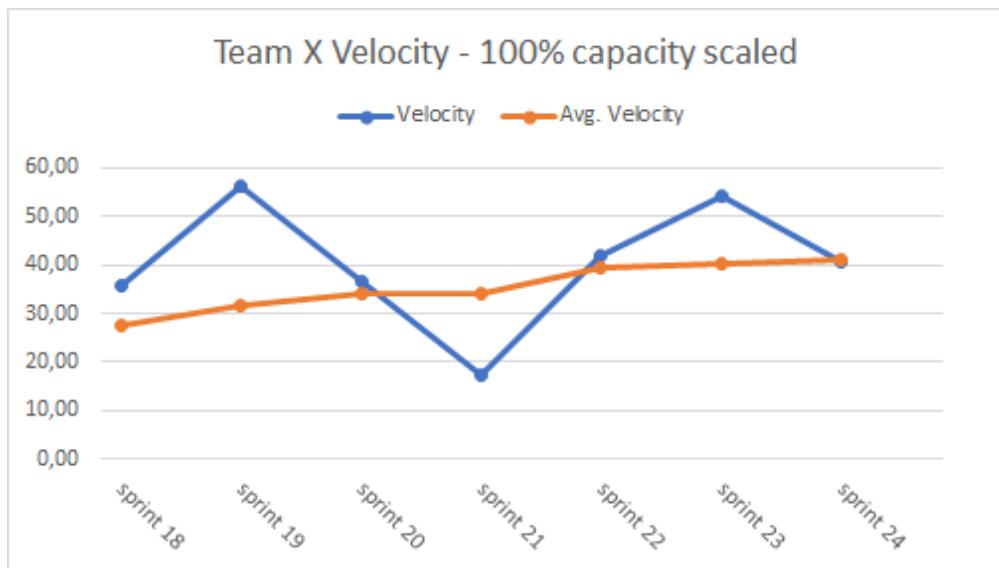
Having Developers with planned hours: 60, 50, 63, 63, 40, 58

The team planned hours = 334 hours

In the end, the planned capacity is 334 / 384 = 87%

Having introduced capacity as another metric next to velocity, this introduces 2 additional metrics based on the combination of both:

- **100% scaled velocity**
- **100% scaled average velocity**



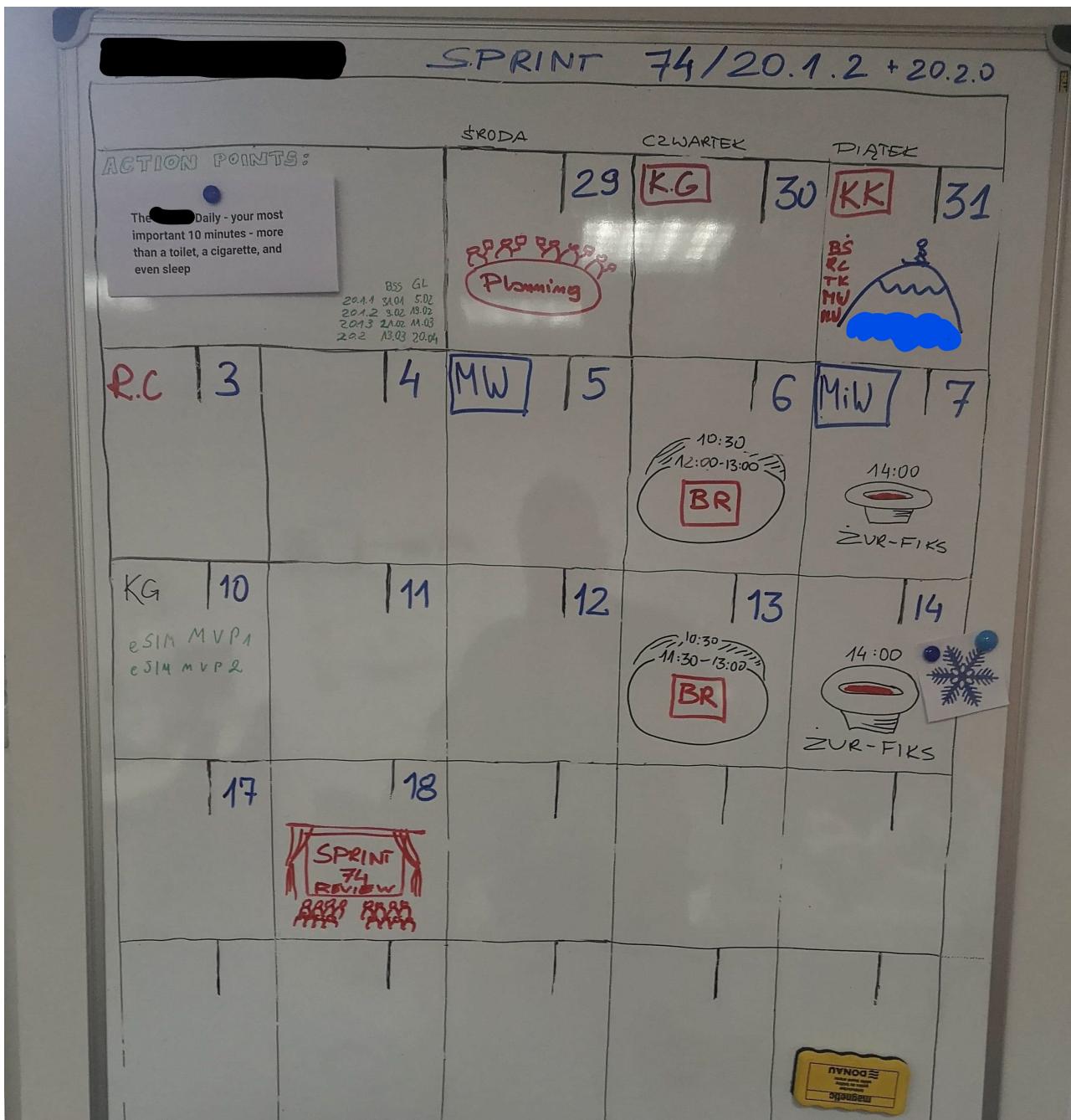
Looking at the sprint and the average velocities through the context of team capacity introduce an additional level of insight into the team's performance and planning. While regular velocity metrics may disregard a potential absence of a large part of the Developers (for example: due to training or sickness) the scaled velocities show us the whole truth, which may impact the decision-making process based on the acquired information.

At the same time, looking from a Product Owner role perspective, the scaled velocity is irrelevant, because what counts is the actual velocity of the team - not what it could be if everyone was available 100% of the time. All in all, all 4 metrics provide valuable information, some of which may be more relevant to specific roles than the other.

It's good practice for the team to regularly declare their capacity for at least the 2-3 upcoming sprints ahead of time (or more if only possible). This gives the Product Owner a better impact view on the upcoming roadmap. At the beginning of every sprint (before Sprint Planning or during it) the Developers should confirm their planned capacity for the sprint. After the sprint, it is also good to reflect and verify the planned capacity in order to maintain proper historical data for evaluating onward plans.

5.7.2.1 Absence planning

Apart from maintaining the team's sprint capacity on a regular basis, it is highly recommended to keep a team members' availability calendar. While team availability is one of the capacity factors, the purpose is slightly different. Having a simple calendar for the team (either in Confluence, or on a big board in the team's physical space for co-located teams) which visualizes the sprint including key events, absences, and major milestones, is an extremely helpful source of information. Images are much more assimilable than words and text, thus having a graphical, illustrated team calendar is an indispensable tool for maintaining internal team awareness. This results in higher predictability of events, a higher chance of risk mitigation, more effective collaboration and all in all, higher productivity of the team.



i Whiteboard Team Calendar

5.7.3 Effective work time

It is common knowledge that no employee can remain fully effective, focused and productive 8 out of 8 daily hours of work. Even set aside efficiency/productivity factors, it is physically impossible to dedicate 8 hours of materialized work in a day, due to basic biological needs. In addition, studies show that an average employee remains productive (with a varying level of effectiveness) between 3 and 6 hours per each workday.

It is important to remain realistic when taking into account the baseline data for calculating our capacities and velocities because every deviation from the actual state makes the result less and less reliable. As shown in the example earlier in the chapter, the effective work time was assumed to be 7 hours; frequently for Scrum Teams, a number **between 6 and 7 hours** is taken into account when we consider a standard workday for an employee when he/she *could* be productive. This ensures that we do not base our planning on the time we actually don't have.

5.7.3.1 Slack time

While effective work time needs to be established and defined for proper evaluation of metrics, slack time is never distinguished from all of the other work of a Scrum Team, even though it may not always be purely related to the current scope of Sprint Backlog.

Slack time is all the time, which is intentionally or unintentionally used for various types of self-management, growth, and improvements within the Developers' work. This can be:

- knowledge transfer, cross-competencies training
- research and experimentation
- reduction of technical debt
- improvement of processes and tools within the team
- other...

Even though *slack time* does not directly contribute to reaching the Sprint Goal at the given moment, it is greatly significant to evolving the team into a high-performing, cross-functional and self-aware organism. Usually, when a Developers complete all of their planned work in the Sprint, before the expiration of the timebox, it discusses with Product Owner the possibilities of pulling in an additional work item which could perhaps be completed within the remaining time. Quite often the amount of time, which is remaining is insufficient to complete anything relevant from the top of the backlog (for example a few remaining hours of the Sprint). That is when the team can become creative and use the time to grow themselves, learn better ways of working, learn new tools or work on minor fixes/tweaks of the product to reduce tech debt. In cases of setups, where multiple teams work on a single product, this is also an opportunity to check up on other teams and see if they are in need of a helping hand - quite frequently this will contribute to delivering the highest value at the end of the sprint, as all of the teams commit to deliver an integrated, potentially releasable Product Increment.

Slack time is not something that should be frowned upon, but encouraged - it fosters self-management within the Developers and contributes to the growth of a high-performing team, which ultimately will pay off.

5.7.4 Boards

Filter-based boards for Developers are standard, supporting tools for tracking and managing team-related tasks in a long-period perspective. Two of the most common boards are **impediments** and **improvements**. Such tasks often are not related (not directly at least) to the Sprint Backlog, therefore they are often forgotten

by the team as the focus on the Sprint Goal intensifies with progressing work. It is *good practice* to keep such things in a visible place for the team - not only to be able to add to the list when issues appear (new ideas pop up all day long) but also to raise awareness about resolving those issues, during the so-called *Slack Time*.

5.7.4.1 Impediments board

Impediments occur all the time. Some of them are addressable by the Developers and some need external assistance to help the team resolve the issue. Some are recurring, while others happen no a singular basis. Of course, it is not necessary nor mandatory to track all of the impediments (raising an issue that can be resolved by the team within minutes is overkill), this habit helps in dealing with long-term, recurring or external impediments. Issues that perhaps may need to be addressed at a higher organizational level, issues that tend to re-appear from time to time (and may need investigation focused more on the root cause) get better visibility on a populated impediments board - indicating that a problem needs to be addressed, otherwise it constitutes to be a risk for the Scrum Team.

5.7.4.2 Improvements board

During a Sprint Retrospective, usually 1 to 3 most significant improvements identified for the team, are added to the top of the next Sprint's Backlog in order to keep focus also on things that help the team grow and improve. Despite that, many more improvements are frequently identified during Sprint Retrospectives (and as many more outside of them) that can bring some benefits for the team. Irrelevant to the fact if there is time to act upon them or not, having a list of such improvements in a visible board raises the chances of addressing it from zero to some.

5.7.5 Small Scale Scrum

All above is applicable in the Small Scale Scrum framework.

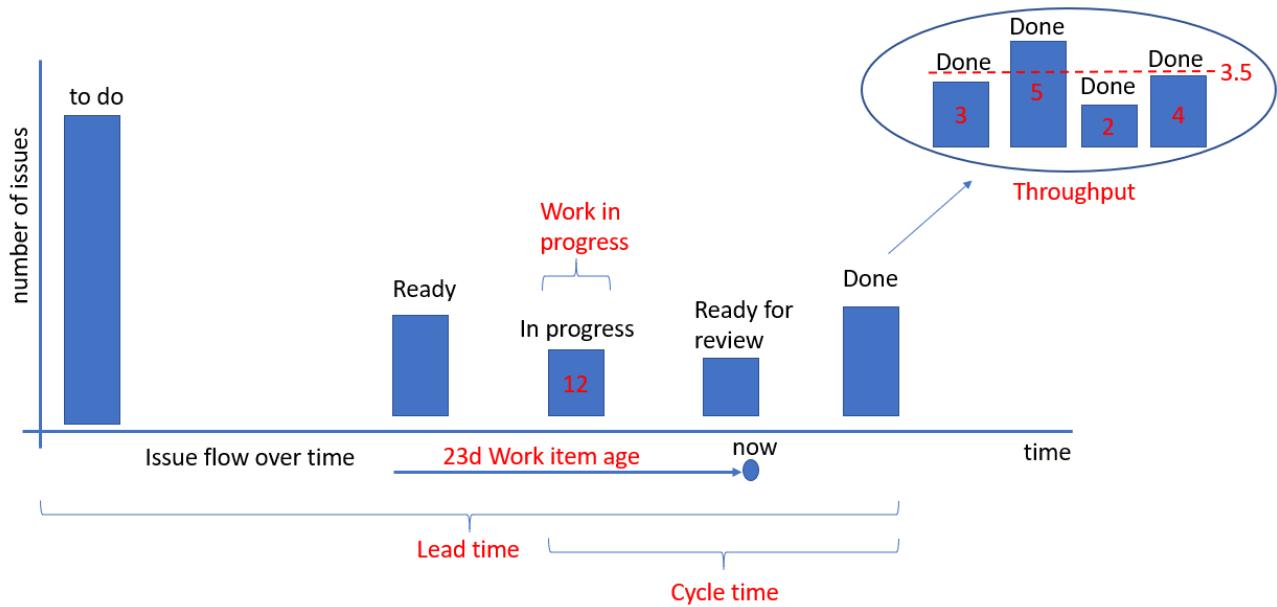
5.7.6 Kanban

Flow management and its optimization are crucial for the Kanban team. There are many metrics that can be used but few of them are especially useful for flow optimization purposes:

- **Work in Progress (WIP)**: The number of work items started but not finished. The team can use the WIP metric to provide transparency about their progress towards reducing their WIP and improving their flow. Note that there is a difference between the WIP metric and the policies a Team uses to limit WIP.
- **Cycle Time**: The amount of elapsed time between when a work item starts and when a work item finishes.
- **Lead Time**: The amount of elapsed time between when the issue has been created and when a work item finishes.
- **Work Item Age**: The amount of time between when a work item started and the current time. This applies only to items that are still in progress.

- **Throughput:** The number of work items finished per unit of time

More about configuring Kanban metrics can be found in this chapter: [5.7.2 - Configuring and using Kanban Team Metrics - Next Generation Agile - Confluence EG A/S \(see page 201\)](#)



For details, please see the relevant article

[4 Key Flow Metrics and how to use them in Scrum's events | Scrum.org²²](#)

three of these metrics (Work in Progress (WIP), Cycle Time, and Throughput) are intrinsically linked by a very straightforward and very powerful relationship known as Little's Law:

...

Little's Law

²² <https://www.scrum.org/resources/blog/4-key-flow-metrics-and-how-use-them-scrums-events>

Little's Law – The Key to Governing Flow

A key tenet governing flow theory is Little's Law, which is a guideline that establishes the following relationship:

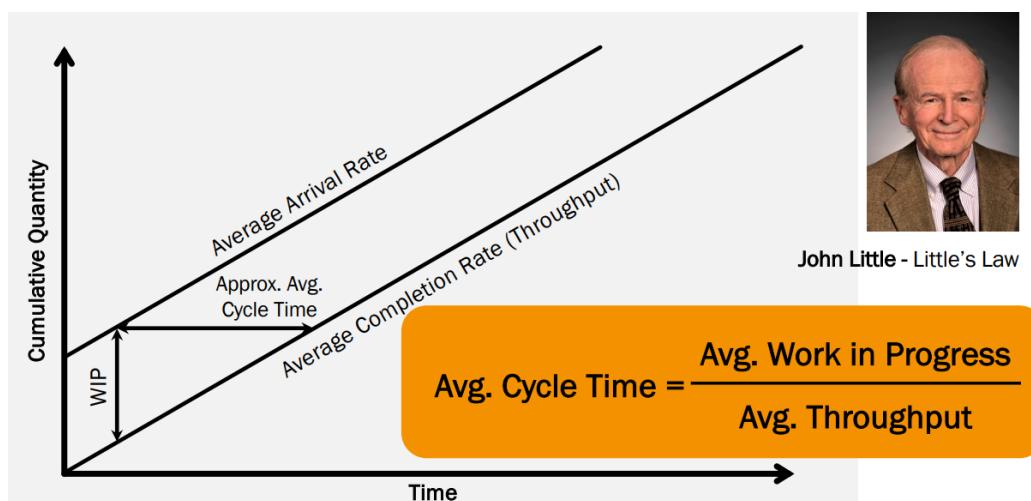
$$\text{average cycle time} = \frac{\text{average work in progress}}{\text{average throughput}}$$

Little's Law reveals that in general, for a given process with a given throughput, the more things that you work on at any given time (on average), the longer it is going to take to finish those things (on average).

If cycle times are too long, the first action Scrum Teams should consider is lowering WIP. Most of the other elements of Kanban are built upon the relationship between WIP and Cycle Time.

Little's Law also shows us how flow theory relies on empiricism by using flow metrics and data to gain transparency into historical flow and then using that data to inform flow inspection and adaptation experiments.

(source scrum.org)



(source scrum.org)

5.7.7 Consultancy teams

All the above is applicable for the Consultancy teams with one remark.

In the situation where:

- the Consultancy team is developing the Product increment under the "customer-funded development" issue category and
- the testing is being done on the customer's side and

- the team has selected the approach where the unfinished user stories are **not** being put to "Done" after handling the validation part to the customer (please refer to [5.4 - The Sprint Backlog \(see page 171\)](#) chapter, Consultancy teams part)

the team's velocity should be interpreted having in mind that these unfinished user stories will negatively affect this metric (issues not finished within the sprint).

This means that:

- we should remember that the development work is finished and all the pending work is unknown (depending on the customer's validation results) and maybe we can take more into the Sprint than the velocity chart suggests;
- it is recommended that we leave some spare capacity in the upcoming sprint so we can handle all the raised bugs from the customer that can come in the Sprint run.

5.7.8 High Product Granularity teams

All above is applicable in the High Product Granularity teams. However, we should put a bit more emphasis on the personal level overview figuring out if the distribution of the issue is balanced across the team. This data can give us a good indication in which areas we should focus on improving the knowledge share avoiding bottlenecks in the future.

5.7.9 5.7.1 - Monte Carlo method. Statistical based forecasting for Scrum and Kanban

5.7.9.1 What is the Monte Carlo simulation

The Monte Carlo simulation is a mathematical technique to make decisions under uncertainty, with usage of statistical model. It runs a large number of random trials using your past data to make prediction for a future time frame.

When we talk about uncertainty, we mean the activity/process, that has more than one possible outcome. A simple example of that kind of activity is flipping a coin. More complex and sophisticated examples might be weather forecasts or hurricane tracking when, we can predict with some level of certainty, possible paths of any tropical storm. The exact path of any hurricane is uncertain, but what meteorologist can do is create sophisticated Monte Carlo simulation models to simulate where the hurricane could go and how likely each of those different paths are.

For software product development processes, it's possible to simulate range of delivery dates, and the probability that comes with each date. For any date in the future, it may use some indicator (for example: the throughput or velocity), to simulate how many work items are likely to get done.

5.7.9.2 Flipping coin example

For better understanding a concept, let's create very simple Monte Carlo simulation model to represent the uncertainty of flipping a coin. Obviously there is a 50% chance of the coin turning up heads and another 50% chance of it turning up tails. We have two, same probable possible outcomes in that case. We don't need to build a Monte Carlo simulation model to gain probability of getting heads or tails for any one coin flip.

But how likely is it to flip a coin five times in a row, and all five times turn up heads? Naturally probability of that event occurring is very low. But how low is it? It's hard to use intuition to answer a question like that. From the theory of probability, it's still a very easy to solve. If each flip of a coin has a 50% chance of turning up heads and you want to find probability of getting heads 5 times in a row, the mathematical way to solve this problem is to simply multiply all probabilities together $50\% * 50\% * 50\% * 50\% * 50\% = 3.125\%$.

Let's try to arrive at that same answer of 3.125%, or at least an answer that is close enough for being useful to decision making. We'll build Monte Carlo simulation for that case. A mathematical model reflecting this process (using excel spreadsheet) and execute it thousands of times. We gonna explore the simulated data afterwards.

Please follow steps bellow

1. Open up Microsoft Excel and start with a blank worksheet. Add framework as listed bellow.

- Row 1 keeps track of the trials. Each trial is one iteration of a coin flipping 5 times in a row. Lets use 1000 trials for this simulation. Number few cells in a row 1 and drag those cells till 1000 trials (ALM column).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	Trial:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	1st	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	0	1
3	2nd	1	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	1
4	3rd	0	0	0	0	0	1	0	0	1	1	0	1	1	0	1	0	0
5	4th	1	1	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0
6	5th	1	1	1	0	0	0	1	0	1	0	1	1	1	0	1	0	0
7	Total:	3	3	2	1	0	3	2	2	4	2	3	3	3	1	2	0	2

- In Cell B2, we're going to build coin flip model by using RANDBETWEEN function. Excel will randomly choose between set of numbers that you give to it. In our case the bottom number will be 0 (that represents a tails) and top will be 1 (which represents a heads).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	Trial:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	1st	=RANDBETWEEN(0,1)		0	0	1	0	1	1	1	0	0	0	0	0	0	0	1
3	2nd	RANDBETWEEN(bottom, top)		1	0	1	0	1	0	0	1	0	1	0	0	0	0	1
4	3rd	0	0	0	0	0	1	0	0	1	1	0	1	1	0	1	0	0
5	4th	1	1	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0
6	5th	1	1	1	0	0	0	1	0	1	0	1	1	1	0	1	0	0
7	Total:	4	3	2	1	0	3	2	2	4	2	3	3	3	1	2	0	2
8																		
9																		
10																		

- Let's copy this function by dragging B2 till B6. Next copy B2B6 selection for each trial by dragging it 1000 columns

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	Trial:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	1st	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	0	1
3	2nd	1	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	1
4	3rd	0	0	0	0	0	1	0	0	1	1	0	1	1	0	1	0	0
5	4th	1	0	0	0	0	1	0	1	0	1	0	1	1	0	1	0	0
6	5th	1	1	0	0	0	0	1	0	1	0	1	1	1	0	1	0	0
7	Total:	3	3	2	1	0	3	2	2	4	2	3	3	3	1	2	0	2
8																		
9																		
10																		

- Let's count total values with sum function with B2 and B6 range. Let's copy it with 1000 tails respectively.

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	Trial:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	1st	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1
3	2nd	1	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1
4	3rd	0	0	0	0	0	1	0	0	1	1	0	1	1	0	1	0	0
5	4th	1	1	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0
6	5th	1	1	1	0	0	0	1	0	1	0	1	1	1	0	1	0	0
7	Total:	=SUM(B2:B6)	2	1	0	3	2	2	4	2	3	3	3	1	2	0	2	
8		SUM(number1, [number2], ...)																
9		26																
10		2.60%																

- Let's use countif function to find out how many times did we flip a coin 5 times in a row and it all turned up 5. The range will be B7 to ALM7 and we want to find out when it's equal to 5.

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	Trial:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	1st	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	0	1
3	2nd	1	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	1
4	3rd	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1	0	0
5	4th	1	1	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0
6	5th	1	1	1	0	0	0	1	0	1	0	1	1	1	0	1	0	0
7	Total:	=COUNTIF(B7:ALM7,"=5")	3	3	2	1	0	3	2	2	4	2	3	3	3	1	2	0
8		COUNTIF(range, criteria)																
9		26																
10		=B9/1000																

- In my case, 26 times out of 1000 the coin flipped heads 5 times in a row. Your results might be different because of RANDBETWEEN function. Actually this number will change each calculation and that's ok 😊. If we want to find probability we just need to take B9 value and divide it by the 1000 trails and format it as an percentage.

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	Trial:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	1st	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	0	1
3	2nd	1	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	1
4	3rd	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1	0	0
5	4th	1	1	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0
6	5th	1	1	1	0	0	0	1	0	1	0	1	1	1	0	1	0	0
7	Total:	=B9/1000	3	3	2	1	0	3	2	2	4	2	3	3	3	1	2	0
8																		
9		26																
10		=B9/1000																

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do																		
Cut Copy Paste Format Painter Clipboard Font Alignment Conditional Formatting Styles Cells Insert Delete Format Cells																		
Calibri 11 A A Wrap Text Percentage % Number Format as Table Cell Styles Cells																		
B10																		
	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	Trial:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	1st	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	0	1
3	2nd	1	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	1
4	3rd	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1	0	0
5	4th	1	1	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0
6	5th	1	1	1	0	0	0	1	0	1	0	1	1	1	0	1	0	0
7	Total:	=B9/1000	3	3	2	1	0	3	2	2	4	2	3	3	3	1	2	0
8																		
9		26																
10		2.60%																

- You can recalculate your spreadsheet multiple times and the number will vary, but it'll stay pretty close to about 3,125%. Its going to come pretty close to the true value, which is 3,125%

- in case of troubles with filling excel sheet please find reference file bellow. [coinflip5x.xlsx](https://confluence.eg.dk/download/attachments/172266185/coinflip5x.xlsx?api=v2&modificationDate=1671814136730&version=1)²³

More trials means grater accuracy of Monte Carlo simulation model. This is simple simulation which answers a question "if I flip a coin five times in a row, how likely will the coin turn up "heads" all five times?". By using simulation model like this one we can approximate the true value.

5.7.9.3 Standard deviation

For more complex cases, when we are dealing with some level of uncertainty, we need to define deviation of possible outcomes. It requires brief understanding of standard deviation concept.

In [statistics](#)²⁴, the **standard deviation**²⁵ is a measure of the amount of variation or [dispersion](#)²⁶ of a set of values.^[1]²⁷ A low standard deviation indicates that the values tend to be close to the [mean](#)²⁸ (also called the [expected value](#)²⁹) of the set, while a high standard deviation indicates that the values are spread out over a wider range. The standard deviation of a [random variable](#)³⁰, [sample](#)³¹, [statistical population](#)³², [data set](#)³³, or [probability distribution](#)³⁴ is the [square root](#)³⁵ of its [variance](#)³⁶. (source wikipedia.org). If you would like to explore more details regarding statistical variance dive into links above.

For our purposes is enough to know that **standard deviation is a measure of variation or dispersion of probable outcomes**. When the standard deviation is lower, the variance of probable outcomes is smaller, and likelihood of obtaining an outcome close to the mean is grater. When the standard deviation is higher, the variance of probable outcomes is grater. The chance of getting extreme values away from the mean is more likely to occur.

²³ <https://confluence.eg.dk/download/attachments/172266185/coinflip5x.xlsx?api=v2&modificationDate=1671814136730&version=1>

²⁴ <https://en.wikipedia.org/wiki/Statistics>

²⁵ [https://en.wikipedia.org/wiki/Deviation_\(statistics\)](https://en.wikipedia.org/wiki/Deviation_(statistics))

²⁶ https://en.wikipedia.org/wiki/Statistical_dispersion

²⁷ https://en.wikipedia.org/wiki/Standard_deviation#cite_note-StatNotes-1

²⁸ <https://en.wikipedia.org/wiki/Mean>

²⁹ https://en.wikipedia.org/wiki/Expected_value

³⁰ https://en.wikipedia.org/wiki/Random_variable

³¹ [https://en.wikipedia.org/wiki/Sample_\(statistics\)](https://en.wikipedia.org/wiki/Sample_(statistics))

³² https://en.wikipedia.org/wiki/Statistical_population

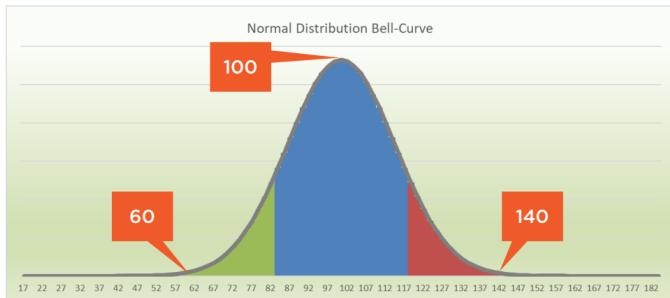
³³ https://en.wikipedia.org/wiki/Data_set

³⁴ https://en.wikipedia.org/wiki/Probability_distribution

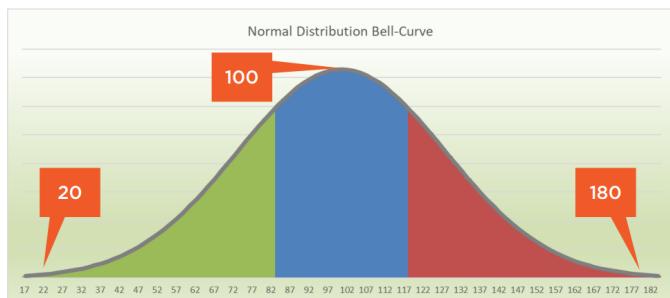
³⁵ https://en.wikipedia.org/wiki/Square_root

³⁶ <https://en.wikipedia.org/wiki/Variance>

In example below we see model of uncertainty that has a mean of 100. When standard deviation is lower the variance of probable outcomes is smaller and likelihood of obtaining an outcome close to the mean is greater. The probability of getting extreme values away from the mean is lower, than when the standard deviation is greater.



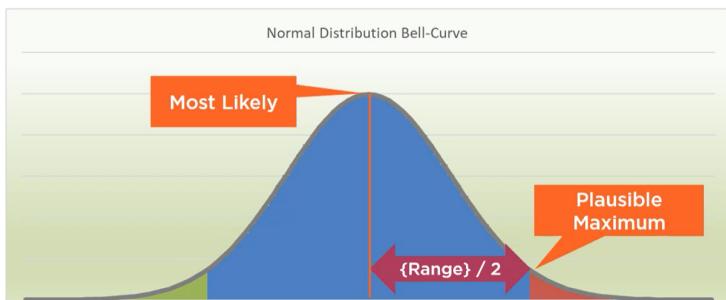
1 Lower Standard Deviation



2 Higher Standard Deviation

5.7.9.3.1 How to reasonable estimate standard deviation?

Using [68–95–99.7 rule³⁷](#). Think about a worst case scenario. The plausible maximum value for how long it might take, assuming normal work variation. Using [68–95–99.7 rule³⁸](#), it's decent practice to set it as two standard deviation away from the mean (95% of results bellow the bell curve). The estimate of standard deviation will be value equal to half the difference between most likely result and plausible unlikely result (worst case).



³⁷ https://en.wikipedia.org/wiki/68–95–99.7_rule

³⁸ https://en.wikipedia.org/wiki/68–95–99.7_rule

SD = (most likely result - plausible worst case result) / 2

Let's take an example of a teams velocity which is estimated to 80 story points. How much variance is there? Let's assume that in worst scenario, team will deliver 40 sp. In this case standard deviation will be $(80 - 40) / 2 = 20$

5.7.9.4 Monte Carlo simulation for Scrum and Kanban forecasting

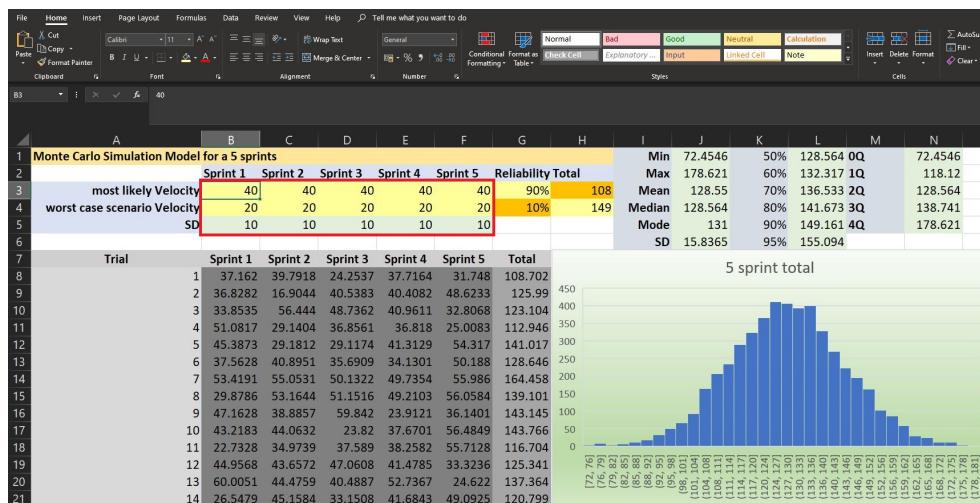
Let's try to apply this knowledge to answer a question: How much work the scrum team might possibly finish in the next five sprints?

Please check bellow example of Monte Carlo model that might be helpful with that.

[Monte Carlo Scrum team.xlsx³⁹](#)

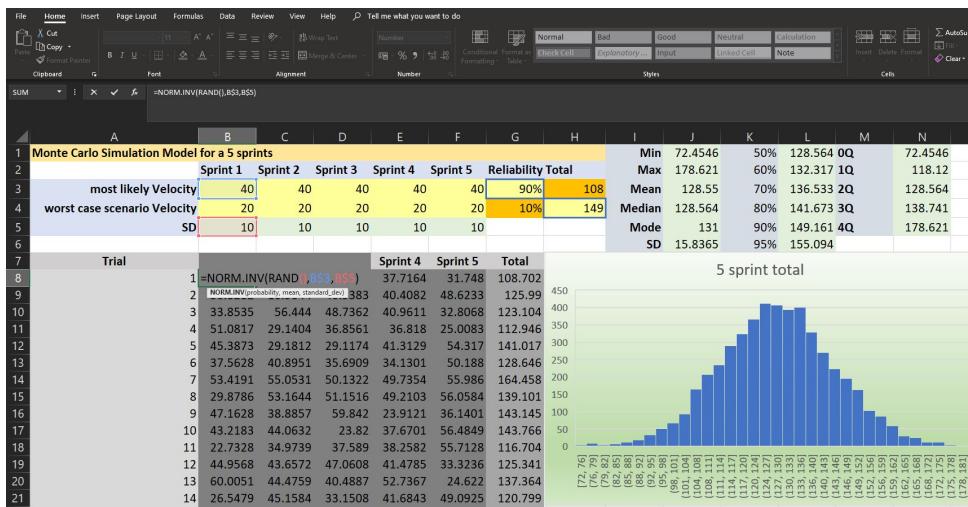
Yellow shade in this model are input cells and need to be filled. Orange share represents calculation output based on input values.

- Row 2 represents headings for each sprint.
- Row 3 contains expected velocity values for the team.
- Row 4 contains worst case scenario for velocity values.
- Row 5 contains standard deviation, which is difference of most likely and worst case scenario divided by 2.



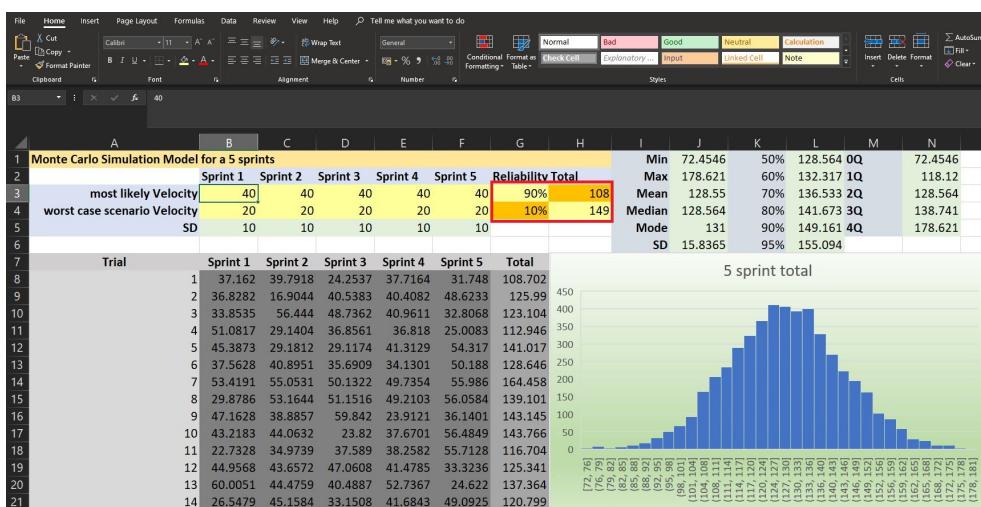
For modeling 5000 trails the NORM.INV function has been used.

³⁹ <https://confluence.eg.dk/download/attachments/172266185/Montecarlo%20Scrum%20team.xlsx?api=v2&modificationDate=1672326922320&version=2>



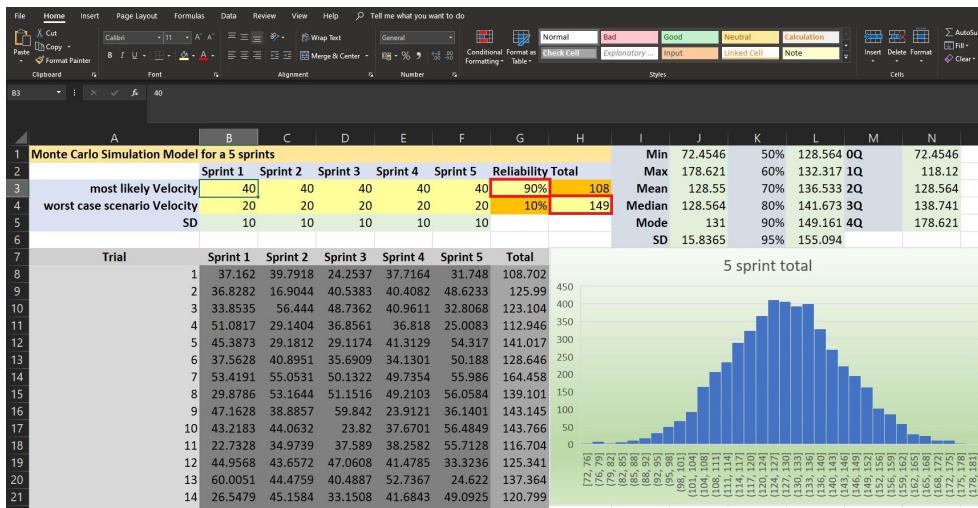
- In normal inverse function the first argument is a probability. In order to get randomness, excel build-in function RAND() should be applied. It will result in a random decimal between 0 and 1. It reflects probability between 0 and 100%.
- Second argument is mean so for Sprint 1 is in cell B3. We need to use relative reference for the column and absolute reference for the row ("B\$3") so we could copy this cell all the way down 5000 times.
- Third argument is standard_deviation. Its in B5 so lets fill it in same way, relative reference for the column and absolute reference for the row ("B\$5")
- After defining function in B8 cell it can be copied by dragging and dropping bottom right corner of the cell up to Sprint 5 column. We have now one trial simulation defined.
- Column G is just a sum of all sprints velocity
- Now we can copy it to desired number of trials (in our case 5000)

5.7.9.4.1 To make probabilistic prediction, how much story points can scrum team get accomplished with some specific reliability take a look at G3:H4 area



- For cell H3 function PERCENTILE.EXC is used. First attribute is array so we need to specify Total values in all trials =PERCENTILE.EXC(G8:G5007,1-G3). Second argument is reliability so in our case it will be G3 cell =PERCENTILE.EXC(G8:G5007,1-G3)
- For cell G4 function PERCENTRANK.EXC is used. In this case we also need same array of all trials and H4 value as a second argument which represents total number of work units. =1-PERCENTRANK.EXC(G8:G5007,H4)

5.7.9.4.2 Forecast example for Scrum team



- Cell G3 represents a reliability.** In this example, it means that (based on input and statistical calculation of 5000 trials) for 90% certainty, scrum team will deliver 108 story points. G3 is editable so you can freely change the value of the cell to adjust reliability to acceptable by the business levels.
- Cell H4 represents total number of story points or other estimation units.** So when PO of stakeholder is asking you what is the chance that scrum team will get at least X number of story points (in our example 149) after 5 sprints, you will get percentage value of probability that event occurring (in our example 10%). H4 is also editable so you can use this calculation to say how likely is delivering specific release estimated for X story points, after 5 sprints.

It's fairly easy to use same model to forecast **Kanban team** delivery. Instead of velocity (in scrum team case) use a **throughput** values as an input. Time equivalent of a sprint in scrum is **cadence** in kanban. Its possible to pull out historical statistics of the team to define throughput. Jira provide lot of (paid) addons but for purpose of counting how many items the team is delivering in specific cadence (for example two weeks), we can build specific query and use it in Jira search issues functionality (Jira/Issues/Search for issues). Please check examples bellow:

```
project = "project code" AND "Team name" = "team name" AND status changed to Done after endOfDay(-14)
BEFORE endOfDay() AND issuetype != Sub-task ORDER BY "whatever (available in JQL syntax) order you prefer"
```

project = NGA AND "Team name" = DevOps AND status changed to Done after endOfDay(-14) BEFORE endOfDay(0) AND issuetype != Sub-task ORDER BY updated ASC

Showing results 1-24 of 24

In this case we see that the team has throughput of 24 issues for last 14 days.

We can also use range for specific dates to reflect team cadence length respectively.

`project = "project code" AND "Team name" = "team name" AND status changed to Done after "start of cadence date" BEFORE "end of cadence date" AND issuetype != Sub-task ORDER BY "whatever (available in JQL syntax) order you prefer"`

project = NGA AND "Team name" = DevOps AND status changed to Done after "2022-12-02" BEFORE "2022-12-16" AND issuetype != Sub-task ORDER BY updated ASC

Showing results 1-19 of 19

Of course throughput of the kanban team can change in time due to many reasons. For instance it can be application of flow management elements (like work in progress limits) or growing maturity of the team. Nevertheless capturing historical stats will help you assess most probable and worst case outcome for future cadences and fill your model with appropriate data respectively.

5.7.10 5.7.2 - Configuring and using Kanban Team Metrics

As stated in the Chapter: [5.7 - Using team metrics - Next Generation Agile - Confluence EG A/S](#) (see page 184) there are 5 main metrics that you can use to track in your Kanban team. Below you will find the information on how to set the Jira reporting and Confluence to track them.

5.7.10.1 Work in progress

The Jira Kanban board is showing directly how many items are in the specific status:

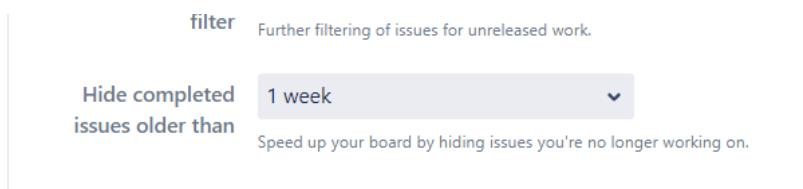
NGA DevOps Kanban board

QUICK FILTERS: Maciek Patryk Piotrek Łukasz Sergii Mateusz Jarek Unassigned

Project: All Type: All Status: All Assignee: All Contains text More Search Clear Filters Advanced

TO DO 190 READ 30 IN PROGRESS 51 DONE 1486

The "Done" column shows you the number of items depending on the board setting showing the items in the given time slot (the available values: 1 week, 2 weeks, 4 weeks). To figure out what your board setting is - go to the "Board" menu in the top right corner, select "Configure" and in the "General" tab at the bottom there will be information about the period the items are visible in:



If needed, you can set the WIP (Work In Progress) limits if the flow of work is not optimal in the Jira Kanban Board configuration.

5.7.10.2 Cycle time

The cycle time metric informs about the amount of elapsed time between when a work item starts and when a work item finishes. You can review the cycle time using the [Control Chart](#)⁴⁰ report in Jira.

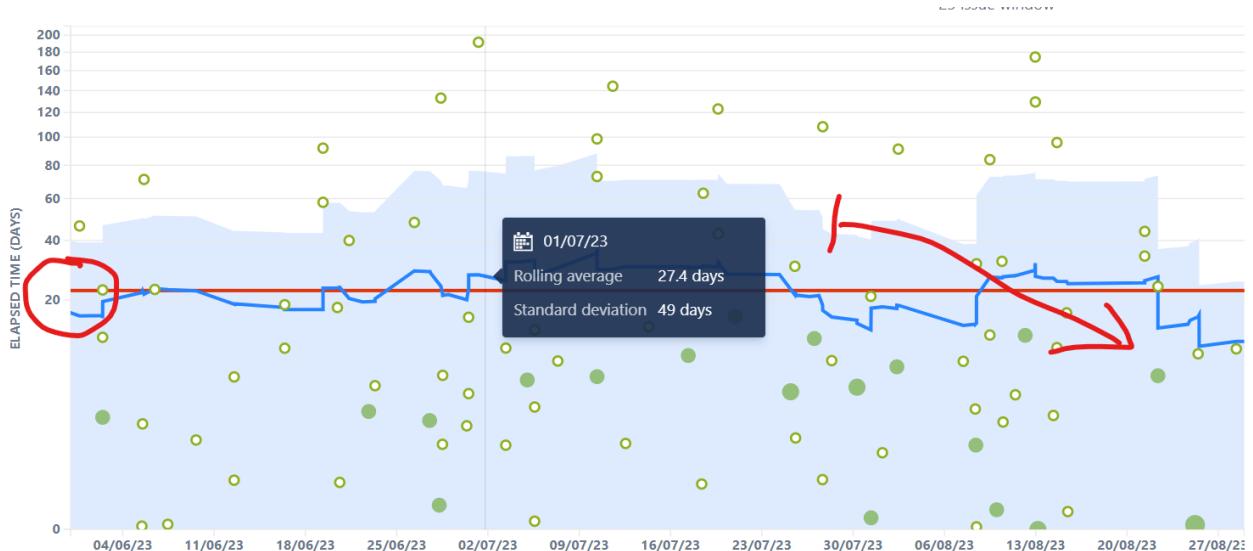
To access the Control Chart go to your Kanban Team board and from the left-side menu select "Reports" select "Control Chart" report:

A screenshot of the Jira Reports page. On the left, there is a sidebar with icons for 'NGA DevOps', 'Kanban board', 'Releases', 'Reports' (which is circled in red), 'Issues', and 'Components'. The main area shows a 'Control Chart' report for 'NGA DevOps'. A dropdown menu is open over the chart, with 'Control Chart' also circled in red. Other options in the dropdown include 'Cumulative Flow Diagram' and 'All reports'. Below the chart, there is a section titled 'How to read this chart' with a brief description and links to 'Learn more' and 'Hide this information'.

At the bottom of the report select all the statuses from "In Progress" till "Done" This list may differ based on the workflow of your Jira project. If needed, you can select the specific time frame that you are interested in:

⁴⁰ <https://support.atlassian.com/jira-software-cloud/docs/view-and-understand-the-control-chart/>

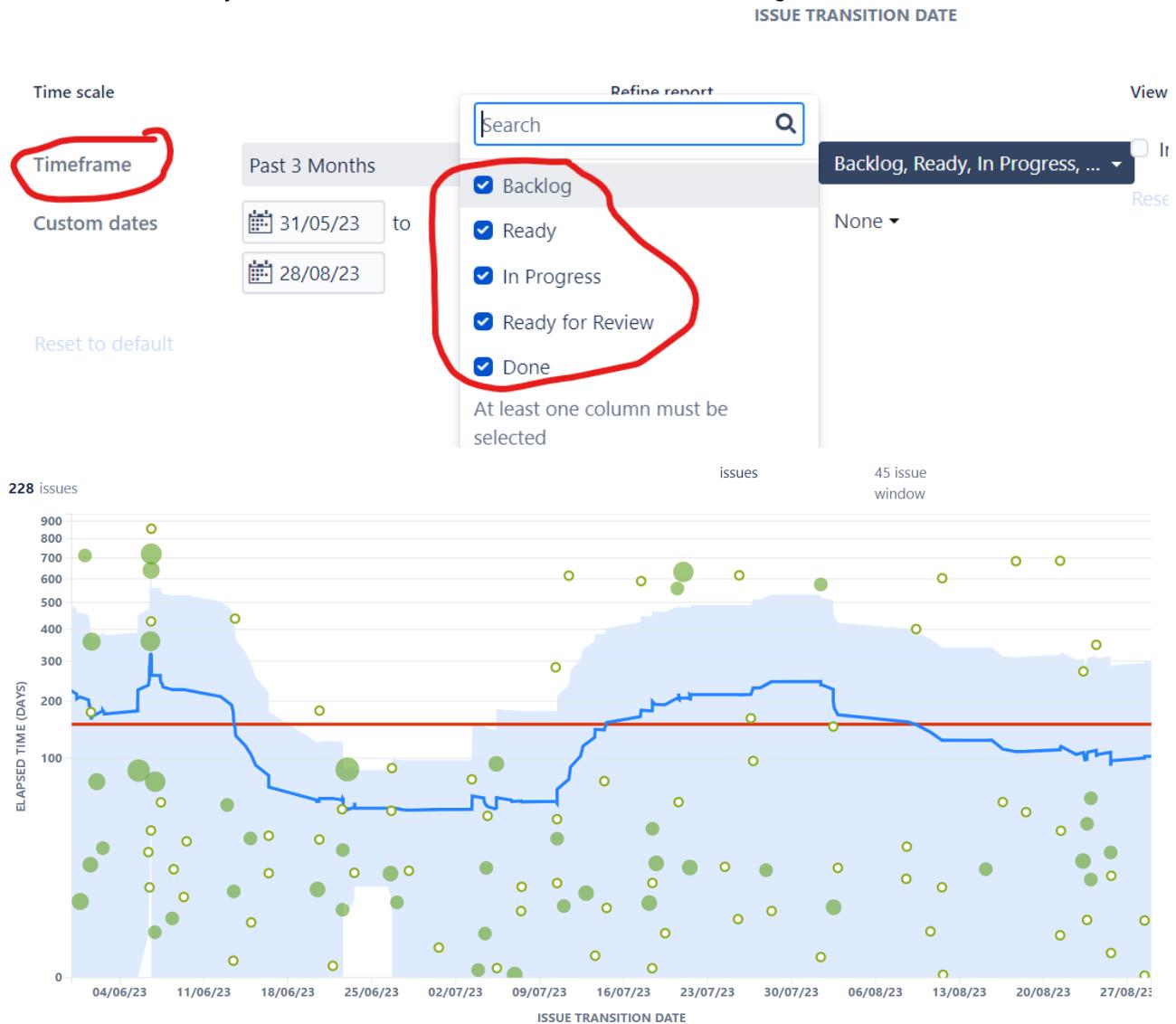
In the below example, you can see that the average time for the completion of the items is about 21-23 days (the red horizontal line) and the rolling average (the blue line) is dropping recently, which means that the team is more efficient and it takes the team less time to complete the work. If the blue line trend is dropping below the red line it means that the team is getting faster with the completion of work, if it goes above, it means that the team is getting slower.



5.7.10.3 Lead Time

The Control Chart in Jira can be used as well to track the Lead Time which shows the amount of elapsed time between when the issue has been created and when a work item finishes.

To set the lead time, you need to select all the statuses in the chart configuration:



i Important

Important: The lead time report is highly dependent on your Backlog health. If you don't do the regular backlog refinement activities and don't delete the Backlog items that are old and will not be developed, then the Lead Time will be significantly longer because these old items will make the average and rolling average longer. This may be the indicator for the team to perform the Backlog cleanup and extended refinement activities.

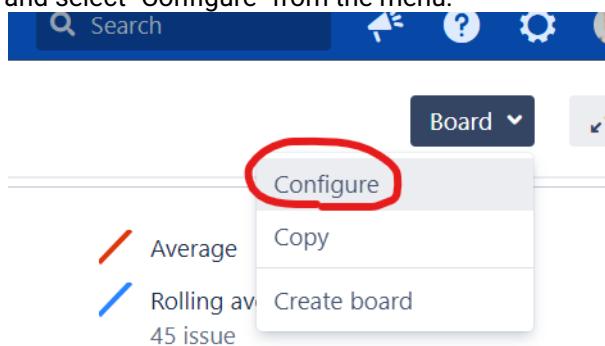
5.7.10.4 Tracking the Lead Time and the Cycle Time for the custom range of data

Jira Control Chart enables you to specify the custom range of data, you would like to track, for example:

- the specific issue types (i.e. Bugs, Stories)
- the specific priority items (i.e. highest)
- a specific team member's work scope

The usage is limited only to the JQL filter you will set. More about configuring JQL filters can be found here: [7.1.10 - Jira JQL hints, useful examples and managing filters - Next Generation Agile - Confluence EG A/S \(see page 293\)](#)

If you want to set the specific data range for the Lead and Cycle time, you need first set the Jira quick filters on your Kanban Board. You need to have board Administrator permissions to progress. Go to the "Board" and select "Configure" from the menu:

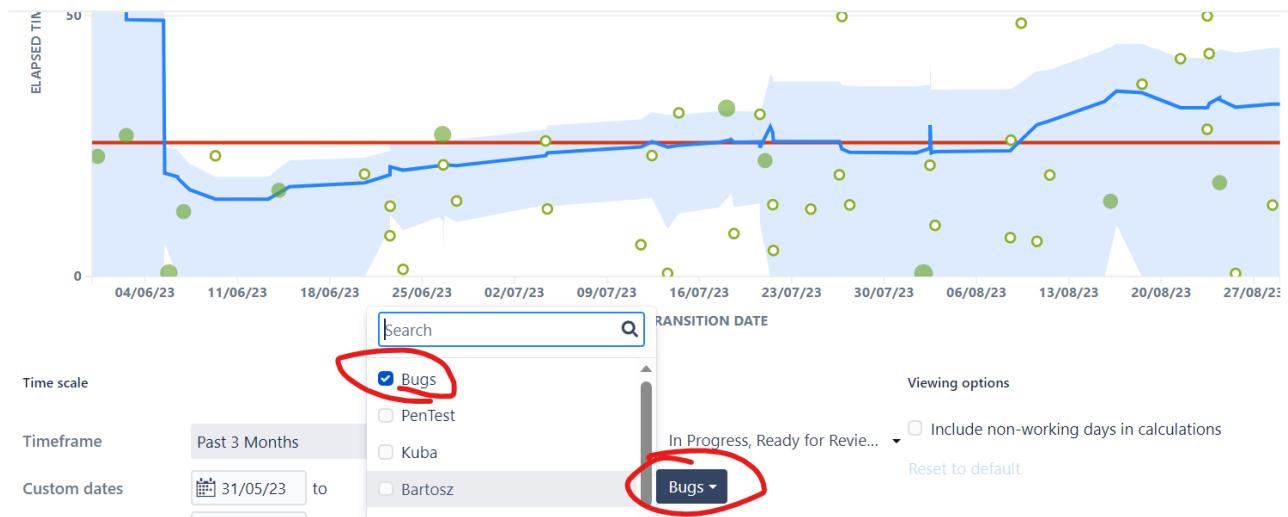


Next, go to the "Quick Filters" tab, specify the filter and add it:

A screenshot of the 'Quick Filters' configuration screen. On the left, a sidebar shows 'CONFIGURATION' with options: General, Columns, Swimlanes, Quick Filters (which is selected and highlighted with a red circle), Card colors, Card layout, and Working days. The main area is titled 'Quick Filters' and contains a table for defining filters. A new filter row is being added, with 'Name' set to 'Bugs' and 'JQL' set to 'issuetype = Bug'. The 'Add' button at the bottom right of the table is circled in red. The table also includes columns for 'Description' and 'Actions'.

After adding the Quick Filter go back to the "Control Chart" report and select the specified quick filter in the report configuration at the bottom and from the drop-down select the respective quick filter:

A screenshot of the 'Control Chart' report configuration. At the top, there is a timeline showing dates from 04/06/23 to 13/08/23. Below the timeline, the section 'ISSUE TRANSITION DATE' is shown. Under 'Time scale', 'Timeframe' is set to 'Past 3 Months' (with '31/05/23' to '28/08/23'). Under 'Viewing options', there is a checkbox for 'Include non-working days in c...' which is unchecked. In the center, there is a 'Refine report' section with 'Columns' set to 'In Progress, Ready for Review...' and a 'Quick Filters' dropdown which is currently set to 'None' (circled in red). Below this, there is a 'Reset to default' link. At the bottom left, there is another 'Reset to default' link.

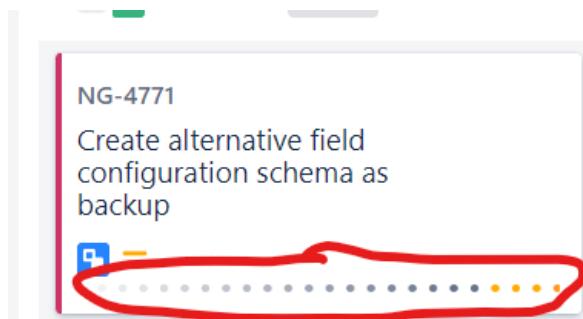


Now the Control Chart will show you the Lead Time or Cycle Time narrowed only to the range specified by the quick filter (in this example only bugs).

This is a very useful feature that can give you a lot of custom information you need.

5.7.10.5 Work item age

The Work Item Age metric shows the amount of time between when a work item started and the current time. This applies only to items that are still in progress. In the Kanban setup, Jira is using the "Days in column" feature.



Days in column, the number of days that an issue has been in a column are represented by a series of dots on the card itself. This helps you see issues that are stagnating – this is particularly useful when your board is displayed as a wallboard.

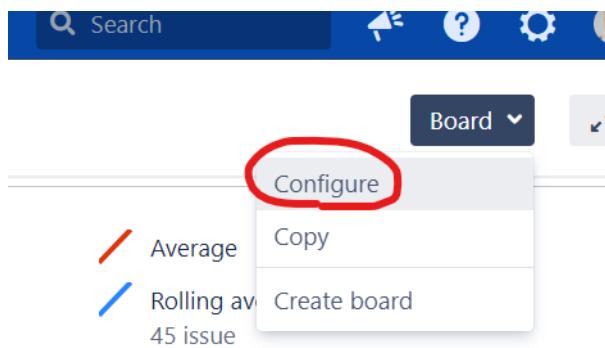
Dots in cards	Number of days in column
•	1 day
..	2 days

Dots in cards	Number of days in column
...	3 days
.....	5 days
.....	8 days
.....	12 days
.....	20 or more days

Note that if you move an issue back to a column where it's previously been, the indicator gives you the cumulative number of days the issue has stayed in that column.

Expressing the items staying too long in the given status

If the Work Item Age is not enough for your team, and you want to exclusively indicate the items that are staying too long in a specific column, you can use the Jira card colors indicator. To do so, go to the "Board" on the top right corner and select "Configure". You need to have board Administrator permissions to progress.



Next in the left side menu select: "Card colors" and specify your JQL for marking the tickets. On the below example, all the items that are staying longer than one week in the "In progress" column will be marked red.

CONFIGURATION

- General
- Columns
- Swimlanes
- Quick Filters
- Card colors**
- Card layout
- Working days
- Issue Detail View
- Agile Tools & Filters

Card colors

Choose a method for assigning colors to your cards. If no method is selected, the cards will not have a colored edge. Any changes to the color configuration for a method will be applied immediately, so you can switch back to it later if you wish.

[Learn more about card colors.](#)

Colors based on: **Queries**

Color	JQL
	status = "In Progress" and status changed BEFORE -1w

[Add](#)

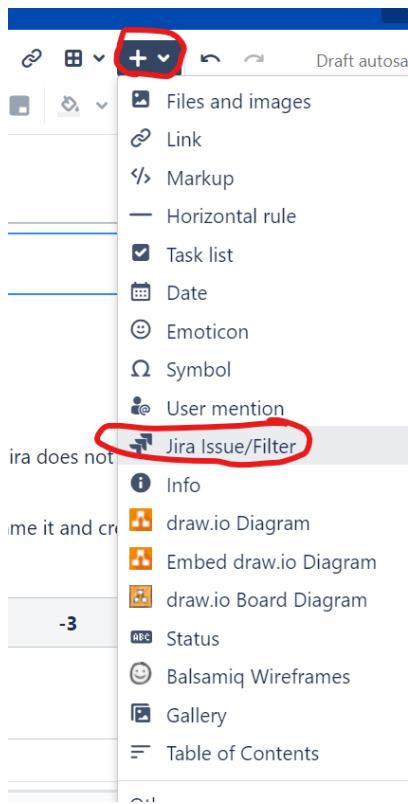
5.7.10.6 Throughput

Throughput is used to express the number of work items finished per unit of time. Unfortunately, Jira does not provide any metric to track throughput, but it can be configured in the Confluence space. The instructions on how to do it can be found below.

Go to the project Confluence space and under the given team page create the additional page. Name it and create the below table:

Week	Current	-1	-2	-3	-4
Throughput (items moved to "Done" in the given week)					

In the each column from current to "-4" add the "Jira Issue Filter add-on:



The double click on each add-on and configure it as follows:

Add the respective JQL finger in the search bar:

For column "Current":

project = [your project name] and "Team name" = [your team name] and status changed to done during (Startofweek(),endofweek())

then press the search magnifying eye icon and under the display options select "Total issues count". Press insert to finalize.

Insert Jira Issue/Filter

Project = NG and "Team name" = DevOps and status changed to done during (Startofweek(-1),endofweek(-1))

Key Summary

Display options

Total issue count

Single issue

Table

Maximum issues

Leave empty to get all issues.

Columns to display

Key	Summary	Issue Type	Created	Updated
Due Date	Assignee	Reporter	Priority	Status
Resolution				

elect Macro Hint: type "Ctrl+Shift+J" in the editor to quickly access this dialog.

Insert **Cancel**

Repeat the above operation with the rest of the columns providing the respective JQL filters:

For column "-1":

project = [your project name] and "Team name" = [your team name] and status changed to done during (Startofweek(-1),endofweek(-1))

For Column "-2":

project = [your project name] and "Team name" = [your team name] and status changed to done during (Startofweek(-2),endofweek(-2))

For Column "-3":

project = [your project name] and "Team name" = [your team name] and status changed to done during (Startofweek(-3),endofweek(-3))

For Column "-4":

project = [your project name] and "Team name" = [your team name] and status changed to done during (Startofweek(-4),endofweek(-4))

When all the above is performed properly, you should get similar results as in the below example:

Throughput:

Week	Current	-1	-2	-3	-4
Throughput (items moved to done in the given week)	1 issue	18 issues	4 issues	30 issues	8 issues

Throughput is a metric for how much work, or progress, you deliver in a specific time period. Throughput doesn't count any unfinished work. Throughput can be used to estimate the delivery of the projected scope of work based on historical performance. With the Throughput, you can measure the future predictability. It can inform you as well about the stability of the work you deliver.

5.8 5.8 - Team page

The Team Page is a common space where the particular Scrum teams can store, track and update all the team:

- information,
- agreements,
- rules,
- activities,
- notes,
- reports,
- etc.

that are relevant and valuable to the Scrum Team. A well ordered and up to date page is a very helpful space to have all team-related topics and subjects in one place, transparent and available to every team member. It improves team collaboration, productivity, and needs for information and knowledge.

In EG are using Confluence, a documentation and collaboration system in which the team pages will be created and maintained. At the beginning of the Agile transformation process, each team will get a Confluence Team Page with its default setup. This page composition is not the target solution – it is the starting point. If the team needs to add or adjust the page content to the particular requirements – it's free to do so within the agreed boundaries, by adding or expanding the existing information. It is recommended to have all the changes agreed within the team so it is clear to all how to use and update the agreed content.

Below you will find the particular Team Page components and short explanation and suggestions under each of them.

5.8.1 The Team Page composition

5.8.1.1 The Team members, roles and responsibilities

One of the first information is the team members list and the scrum roles assigned to the particular persons. It is good to share also the mail addresses and pictures to improve identification and contact. This can allow the stakeholders to contact the right persons when needed. The team page is also valuable to the new team

members or to other Scrum teams working under the same product. Below you can find the example of team members list:

The Team

	Mathias Hansen mafha@eg.dk	Product Owner
	Birgit Egesbæk biege@eg.dk	Senior Developer
	Jan Tovgaard jatov@eg.dk	Senior Developer
	Line Nielsen libni@eg.dk	Senior Developer
	Brian Nielsen brbni@eg.dk	Senior Developer and Scrum Master
	Johnny Madsen johma@eg.dk	Senior Developer

5.8.2 Team rules, agreements, Definition of Ready, Definition of Done

5.8.2.1 Team working agreement

When different personalities are starting to work together, team members begin to learn about each other's approach to work, strong and weak points, skillsets and behaviors. Sooner or later the first arguments around how to cooperate occur. To prevent or limit those arguments and to set up the common understanding and approach to the Team cooperation the Team working agreement should be drafted, discussed and accepted. The following areas should be taken into consideration when setting up the agreement:

- Team values (i.e. openness, courage, professionalism or kindness)
- Behaviors (i.e. "if you need help just ask and the team will help you" or "please be punctual, instead of sending the mail to your colleague just discuss the topic" or "we are sharing the knowledge");
- Meeting details (i.e. "no phones at the meetings", "no broad technical discussion at Daily Scrum", "we are drafting meeting notes with follow up plans");
- Technical practices (i.e. "to delete feature branches when the code is merged to Master" or "write tests for new features" or "to code reusable components always if possible")
- Communication tools - (i.e. "we are using MS Teams when communicating with other team members" or "first we speak then we write emails")
- Any other subjects or areas the Scrum Team should cover and set up a common approach.

The team working agreement should be discussed and accepted by all the team members. It is good to validate the agreement from time to time and update it when the Team gets mature and always when a new way of common team approach occurs.

Why is it important to have a written agreement instead of just discussed a set of rules? This is because unwritten rules have a tendency to get ignored, forgotten or be understood differently by various team members. Also, they are not visible to the new team members and cannot be reviewed and improved.

5.8.2.2 Definition of Ready and Definition of Done

The Team page is also a default place where the team is keeping the agreed Definition of Ready and Definition of Done. Usually, both of them are published in a way of the checklists and are different for the particular issue type. They should be reviewed with each item in order to make them Ready or Done.

It is good to have the date when the current Definition of Ready and Definition of Done has been last time agreed and changed. This can give the Team input when it was the last time reviewed and if we should maybe do the update. Changing both the above definitions should not be done without whole Scrum Team's discussion and agreement. You have to also remember that there is a default EG Definition of Ready and Done baseline which is common to all EG Scrum Teams and should also not be changed. The team may build on top of baseline DoR and DoD for EG, as well as on top of the Business Unit's baseline DoR and DoD (if one exists), but may not change the globally binding, existing set of statements.

For more information please read [5.5 - The Definition of Ready \(see page 176\)](#) and [5.6 - The Definition of Done \(see page 180\)](#)

5.8.3 Visualization of team metrics

There are at least two categories of metrics that should be constantly updated and validated by the Team. Those are the Developers capacity and velocity.

5.8.3.1 Capacity

At the beginning of each sprint, the Developers are presented with a set of user stories, tasks, and bugs that are the candidates for the upcoming sprint. The team is estimating those items to determine how much work is needed to make them Done. Capacity planning helps a team understand how many story points or other metrics (i.e. hours) they are likely to accomplish. The Developers' capacity is information about how much work particular team members are capable to take in the upcoming sprint. When estimating the capacity we have to take under consideration the availability of the particular team members (all personal day-offs, sick-offs, company day-offs or bank holidays) and the time they can work within the Sprint (i.e. Tester is sharing 50% of his time between two Scrum Teams; or planned meetings, training, etc.). For more information please review: [5.7 - Using team metrics \(see page 184\)](#)

It is necessary to have all this information stored together and updated before the Sprint Planning meeting for the Developers so at the planning the team is aware of the capacity and can plan the Sprint accordingly. It is usually done by creating a table with particular team members and their availability. It can be also useful to trace if the particular capabilities are well covered. For example, if the tester is available only three days in the two-week sprint and successful testing is an essential part of the Definition of Done, the Team should plan Sprint a bit different compared to 100% default availability.

5.8.3.2 Velocity

Velocity is a simple and powerful metric of how much work has been accomplished during the completed sprint and Done stories within it. It is information about how many story points (or another measurement units) team has completed. This is valuable information about how much workload can be taken into the

future Sprints and how the team is progressing from sprint to sprint (a precious input for retrospective discussion and setting up improvements). For more information please review: [5.7 - Using team metrics \(see page 184\)](#)

It is very important to store velocity charts or graphs on the Team page and update them once per sprint at least prior to the Sprint Planning session. There is a possibility to generate a velocity chart in Jira, which can be used to visualize the velocity (for more information please see: [Velocity report⁴¹](#)); however using other simple tools, like excel is also possible. It is available at the list from the “Reports” side menu. It can be easily moved to the Team page (as a link or as a screenshot to have it visible at the Team Page).

5.8.4 Retrospective notes, sprint reports, other reports

The retrospective notes page is being used for addressing all the topics for an upcoming retrospective meeting that occurs during the sprint. It is also space where the Team is storing all retrospective discussion outputs, agreements and selected points for improvement in the future. For more information please read [4.6 - The Sprint Retrospective \(see page 126\)](#) It is up to the team to arrange the space and store all the retrospective materials. Please review how to create retrospective notes in Confluence: [7.2.2 - Documenting a Sprint Retrospective \(see page 321\)](#)

Any sort of other reports that may be useful for tracking the Team progress can be also stored on the Team page. This can be Sprint reports or burndown or burnup reports available in Jira from the sidebar “Reports” or any other reports that the Team would like to refer to in order to make assumptions, improvements or info sharing.

5.8.4.1 Team events calendar

The team events calendar is a useful source of information about all teams Scrum ceremonies. It can be visualized in the form of a table where particular meetings (day of the meeting, duration, and meeting room or remote meeting details) are available to the team members and for the stakeholders if needed.

5.8.4.2 Team leave calendar

The team leave calendar is information about the absence of all Team members, planned or not planned. It can provide the Team input about any leaves and their duration. It can be a precious input for team capacity and to plan the Sprint well. It can also give the input to the Developers on how to react on unplanned leaves (i.e. sick absence) and to replan the work within the current Sprint. It is essential to update the calendar as soon as new information about the leave occurs.

5.8.4.3 Knowledge base

The Team can also create the Knowledge base space where all the valuable materials about sharing the knowledge can be stored. For example, if one of the team members has training and would like to share the training materials, this is the correct space to do it. This is a place for online materials, videos, links, documents and everything else that would be useful for the team to be better in the future.

⁴¹ <https://confluence.atlassian.com/jirasoftwareserver/velocity-chart-938845700.html>

5.8.5 Who should manage and update the Team page?

It is the whole Scrum Team's responsibility to have the Team Page up to date and managed well. The Team should self-manage and share the responsibility of content management. It is good to agree in advance who will take care of the particular page spaces so it is clear who is responsible for it and the workload is well balanced so there is no single person burden. The Team can also agree on the rotation of responsible persons. Never the less all the Team members should be able and encouraged to update the Team Page always when needed.

5.8.6 The page transparency

The Team page is available to all the Confluence users that have granted access to it. It means, that the team can use the page to show the selected stakeholders all the information the Team finds useful to share. On the other hand, if you need to limit access to the page you can always do so by proper Confluence page configuration. This can be done on two levels:

- a) the first level can limit the editing credentials, which means that everybody can view the page, but the editing is limited to the selected persons;
- b) the second level limits access to the particular confluence page. For example, if the team decides that the retrospective space should be accessible only by the Team members, it can be set up accordingly.

At the top of each Confluence page, there is a padlock where the above limitations can be set up. It depends on particular system user permissions if the above functions are available. If needed please request the needed credentials.

You have to remember that the Team Page is a space where all topics and areas related to the Scrum Team can be created, stored, updated and managed. If the Team feels that it would be valuable to add and cover other areas not mentioned in this chapter it is free to do so. The general rule is that the content should be arranged in a clear manner, up to date and provide the value to the Team.

6 6 - Cross-team & cross-product collaboration

- [6.1 - When we need to reach outside the agile team \(see page 216\)](#)
- [6.2 - Cross-team collaboration in a single product \(see page 219\)](#)
- [6.3 - Managing a single product backlog across multiple teams \(see page 225\)](#)
- [6.4 - Cross-product collaboration by one or more teams \(see page 231\)](#)
- [6.5 - Managing priorities, dependencies and roadmaps cross-product \(see page 236\)](#)

6.1 6.1 - When we need to reach outside the agile team

One of the benefits of Agile frameworks and methods is the emphasis on making things simpler. Developing software products is complex enough, thus the effort to simplify the surrounding processes should be high enough to help the teams focus on building the right, qualitative products.

One of the Scrum values is **Focus**, which brings simplicity into the process by saying that teams should concentrate on creating one thing well at a time. In KanBan, one of the practices is to limit **Work in progress**, which supports the vision that teams cannot deliver high quality when multitasking and switching context regularly.

Having cross-functional, self-managing teams makes the team setup **simple** enough to not worry about that aspect of the product delivery infrastructure, but rather shift the effort towards solving complex problems. However, now more frequently than ever we face the need to enable teams to collaborate on a product or even cross-product level when delivering the best solutions to EG customers.

This chapter focuses on explaining how to utilize the potential of EG agile teams to collaborate and communicate in the scope of a single product or cross-product delivery. The chapter also elaborates on working with multiple products by a single agile team, to best utilize their capacity in a high product granularity setup. The emphasis is on maintaining proper prioritization across such setups and managing dependencies.

6.1.1 The first rule of scaling Agile: Don't scale

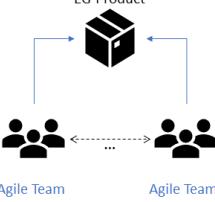
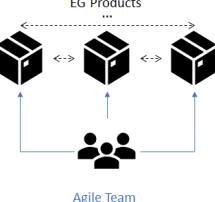
By default, agile teams should be built to be cross-functional, self-managing teams as defined in: [3.1 - What an EG scrum team looks like and how it operates \(see page 47\)](#) to be holistically responsible for delivering increments to the dedicated product. In some cases, however, there might occur a necessity to introduce cross-team or even cross-product collaboration to reach the product goal in the long run, as well as to gain some short-term quick wins.

Before considering the need for, circumstances and the approach to expanding the product delivery team setup into a multi-team and/or multi-product setup, please evaluate if you have exhausted all possibilities of avoiding dependencies between teams. Introducing scaled solutions into your setup introduces dependencies and risk, which in turn impacts the complexity of delivering your to-be qualitative product. Reach out to your Agile Coach and discuss how this could be avoided and if there is another way to resolve the impediment that you are facing.

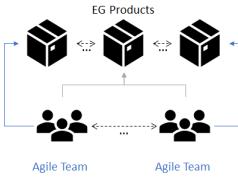
A good example of how to avoid scaling your agile team setup, but acquiring the help/competences you need is reusing components/services across the organization. Apart from saving time and cost, it allows

agile teams to introduce technological concepts into the products which they are building, without compromising their integrity and self-management (more: [Shared Components⁴²](#)).

Having evaluated all other options, when you still consider this to be the way to proceed to gain the most value for your product and customers, please try to assess which situation best describes your requirements/context:

	<p>Single product - multiple teams</p> <p>The simplest scenario, and most likely the most common that might be expected. In such a situation, there is one large product, where the expectations regarding the number of deliverable features and the extensiveness of the product require more capacity to meet the expectations than that of a single team.</p> <p>There may be more than one PO involved in being responsible for various areas of the product and collaborating with the teams, however, the PM should remain one, overlooking the product roadmap and vision. The Product Manager will resolve any conflicts of interest between PO's if occur, being a final decision-maker.</p> <p>Teams can have specialization areas that cover some specific functionalities of the system, however, they should maintain a limited possibility of working on all aspects of the product.</p> <p>It may be helpful for POs and teams to be matched in the specific feature areas of the product to be most efficient.</p>
	<p>Multiple products - single team</p> <p>A scenario representing the usual case of a high-granularity team, which works on more than one product (often several small products). As compared to the previous scenario, here the capacity of the agile team allows progress in developing or maintaining more than a single (usually smaller) product.</p> <p>The synchronization load in this case falls onto the product part, where multiple POs need to align product priorities between each other for the team to follow. A single PM overlooking those several products would definitely be helpful.</p> <p>The setup requires proper planning of work to avoid context switching and increase the focus factor within cycles.</p>

⁴² <https://confluence.eg.dk/display/NG/Shared+Components>

	<p>Multiple products - multiple teams</p> <p>The least preferred scenario usually refers to a (temporary) lack of possibility of establishing proper independent agile teams. It is strongly recommended, to aim to resolve such setup gradually shifting to eliminate the outstanding product-team dependencies.</p> <p>The scenario occurs as well in the situation when the teams from different Products are joining temporary forces to develop a feature, that will be used in both products to achieve the synergy effect.</p> <p>This scenario requires synchronization both on the team level in the context of the affected products, as well as on the product level in the context of the affected teams. It is helpful to establish areas that can be with some effort extracted from the setup thus forming a sub-setup that is more or less independent of the rest. This will reduce the complexity.</p> <p>Limiting the number of PMs who overlook the product roadmaps, and having a good relation PO - team setup will help reduce the risk of issues resulting from unresolved or unidentified dependencies.</p>
---	---

6.1.2 The main benefits of extending your team-product setup

- allows using the potential of multiple agile teams to gain delivery rate increase, especially in achieving the MVP in a rigorous time frame
- can support temporary ramp-up periods of newly formed agile teams
- can help to optimize the use of the capacity of a mature agile team, especially supporting multiple products
- allows better management of dependencies and risks, when offering complex product solutions to the customer

6.1.3 The main drawbacks of extending your team-product setup

- raised overall complexity in delivering EG products, introducing risk to end quality
- dependencies on team levels, resulting in the need to synchronize between teams and manage the dependencies (time)
- dependencies on product (backlog) levels, resulting in the need to synchronize between PO/PM roles and manage those dependencies (time)
- the need for additional synchronization on PO, PM role level and maintaining cross-product (per team) prioritization with respect to the product roadmap and vision
- extending time-to-market in some cases, where proper implementation is not achieved
- delayed feedback loop
- collaboration challenges in cases, where the teams are using a different framework or its variations

6.1.4 What is not recommended - try to avoid

- **sharing people between multiple teams** - every team follows their cycle, whether it's Scrum, KanBan, or another approach, they live by a certain pattern of events and actions to plan, execute and review their work. Each and every group of people also becomes a true team over time, building a relationship within that group and becoming an efficient and high-performing entity. Having people who must be part of more than one team, impedes the growth of all the teams affected, as well as their cycle stability and in the end their performance. People in multiple teams should theoretically participate in the events cycle for all teams they are a part of, which can become a bottleneck for both sides.
- **thinking about how to scale rather than if you need to scale, or what to do not to scale** - as stated at the beginning, scaling introduces complexity to the process, which can, in turn, impact the well-being of the end product. Before starting to expand your setup, investigate how it could be otherwise avoided.
- **adding more teams to boost the capacity** - of course increasing the size or number of teams can speed up development to some point, but it also is important to know that this increase is not linear and will decline over a certain team setup size. It can be more effective and cost-efficient, to look for ways to optimize and support the team's work to become more productive and reduce waste in their process, rather than adding new teams to reach the deadline quicker.

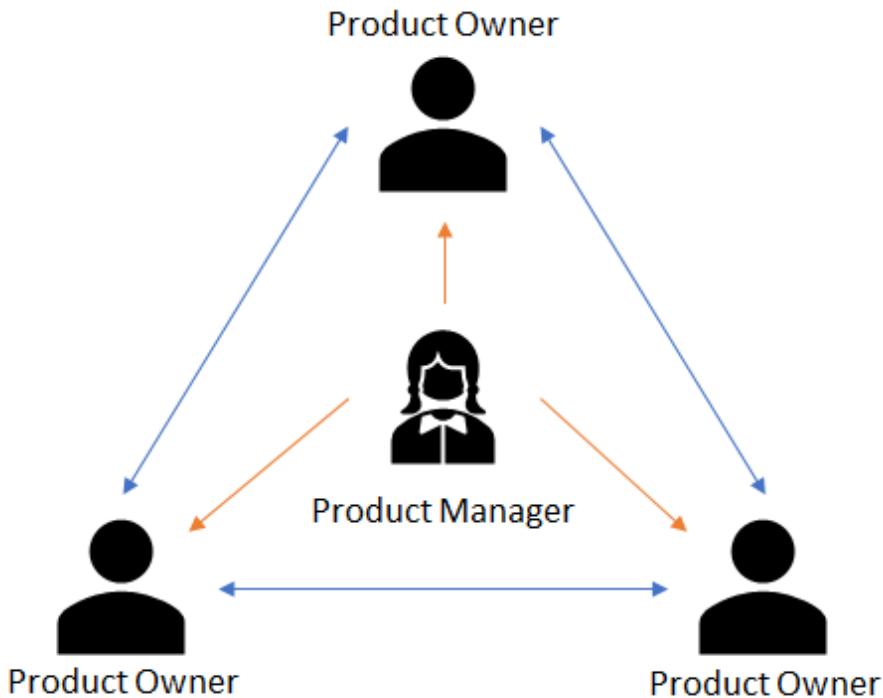
6.2 6.2 - Cross-team collaboration in a single product

Some products, due to their complex nature or broad scope, require work to be done by a significant number of people; much more than can make up for a single Agile Team. In such cases, multiple, cross-functional Agile Teams are formed in order to work on delivering business value for the products, in their sprints with a resulting **one** and **consistent** potentially releasable Product Increment at the end.

Having multiple Agile Teams working on a single Product Backlog introduces some complexity to the self-management of those teams and additional points of synchronization need to be introduced in order to cope with the distributed work in progress and cross-team dependencies. The process described here applies best to up to 3 teams and should there be a need for a higher-scaling model, it is recommended to discuss it individually with the Agile Coach.

6.2.1 Product Owner hierarchy

As mentioned in [3.1 - What an EG scrum team looks like and how it operates \(see page 47\)](#) 2 small Teams can share a Product Owner and Scrum Master. However it is also possible to have 2 teams with dedicated Scrum Masters and Product Owners working on 1 Product Backlog. In such a scenario, when we have more than 1 PO resulting from multiple small or regular Scrum Teams a hierarchy needs to be established to enforce Product Owner empowerment and decisiveness.



- 1 Product and hence 1 high-level Product Backlog should be controlled by only 1 Product Manager, who has the ultimate say in the decision-making process
 - in an undesirable situation, where the product extensity requires more than 1 person operating on a Product Manager level, a head Product Manager should be indicated to have final say in all challenging decisions
- all of the Product Owners collaborate with each other to align the Product Backlog prioritization among the teams and to synchronize common areas of the product across the teams, to reach a consistent Product Increment
- only if an agreement between the Product Owners cannot be reached on topic regarding the Product and its Backlog, they should address the issue to the Product Manager who has final say on the subject and makes a binding decision

6.2.2 Work distribution and planning synchronization

Working with multiple teams on a single product, hence a single Product Backlog, requires much more organization than in case of a single team - single backlog scenario, as defined in the scope of the EG Agile Playbook (more in: [6.3 - Managing a single product backlog across multiple teams \(see page 225\)](#)). In order to conduct valuable backlog refinement sessions, as well as proper Sprint Planning by all participating teams, an additional cross-team meeting needs to take place.

The meeting, called *Pre-Planning* should be conducted as a **1-hour meeting per (2 week) Sprint**. The meeting should take place at a regular location and time, preferably at the beginning of each sprint before any backlog refinement sessions take place.

The participants of the *Pre-Planning* meeting include:

- at least 2 representatives from every Scrum Team working on the single product (more if deemed necessary by the Developers)
- Scrum Masters of all teams working on the single product
- Product Owners of all teams working on the single product
- (optional) Product Manager (if deemed necessary by the Product Owners)

The goal of the *Pre-Planning* meeting is:

- to identify the candidate backlog items for the upcoming sprint(s)
- to draw a high-level (Sprint focused) timeline regarding the identified candidate backlog items
- to create a distribution of candidate backlog items across teams, based on input from the Developers' representatives and the prioritized Product Backlog
- to eliminate or identify key, high-level dependencies between teams and address them

This meeting is a mandatory pre-requisite to following backlog refinement sessions and Sprint Planning events for the teams. At the beginning, the meeting will address usually the upcoming sprint only, but with time the *Pre-Planning* can look forward to as many as 3 next Sprints in synchronization with the backlog refinements being conducted by the participating teams.

6.2.2.1 Additional Pre-Planning session

If found necessary by the collaborating teams, it is possible to conduct an additional preparatory *Pre-Planning* session during the same Sprint. The meeting should last **no longer than 30 minutes** and should take place at the end of the sprint, after the Sprint Review and before the Sprint Planning of the next sprint.

The meeting should focus solely on the next Sprint with Sprint Planning in mind. The purpose of the meeting is to introduce even more clarify on the planned scope of the team Sprint Backlog working across the single product. The participants of the meeting may be limited to only those teams, which are affected by the requirement for the additional meeting.

6.2.2.2 Backlog Refinement

Having an up-to-date Product Backlog with agreed, distributed cross-team backlog item candidates for at least the upcoming sprint, it is possible to proceed with backlog refinement sessions on a team level as described in [4.2 - The Backlog Refinement \(see page 113\)](#). It is also possible to have representatives from other teams attend the backlog refinement session of another team on a need-to basis.

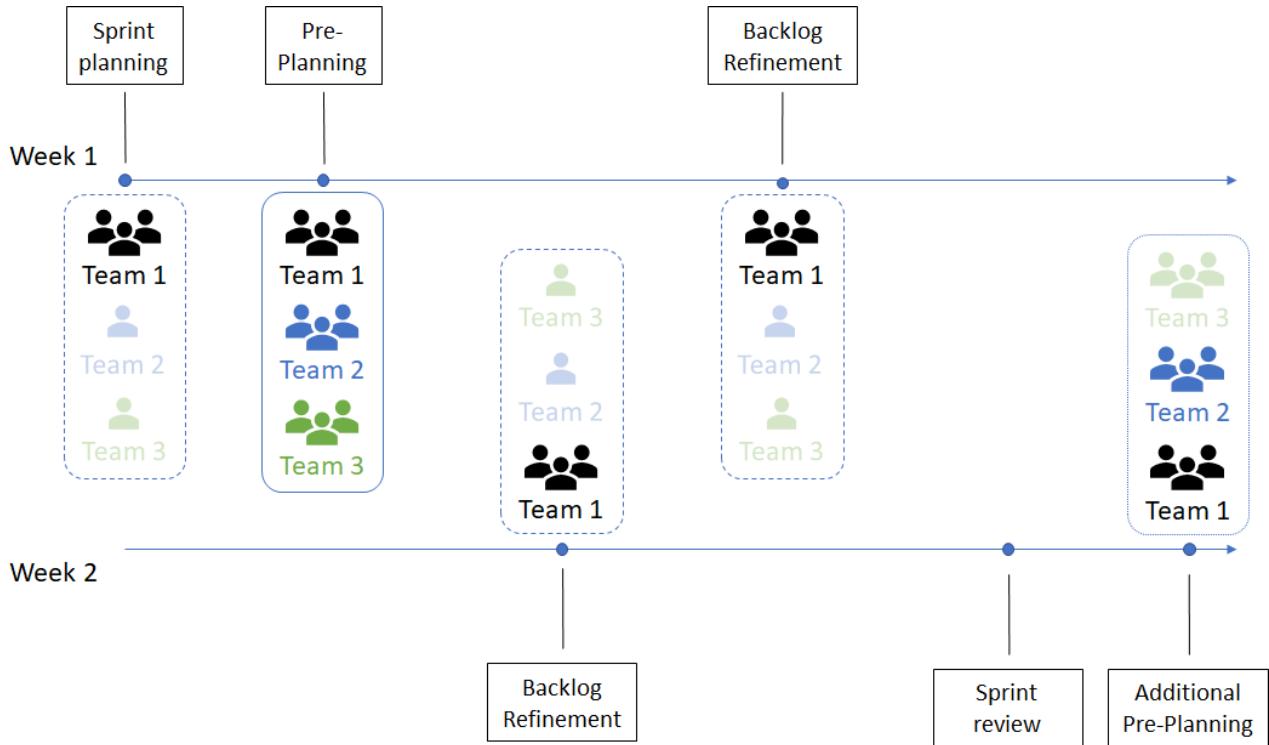
This could potentially occur when:

- a high dependency exists between teams' candidate backlog items
- one of the refined Product Backlog items is of high relevance to the overall product

6.2.2.3 Sprint Planning

Similarly to the backlog refinement session, with an agreed distribution of product backlog candidates across the teams, it is possible to conduct the Sprint Planning on a team level as for a single team configuration (described in [4.3 - The Sprint Planning \(see page 117\)](#)). An additional *Pre-Planning* meeting at the end of the sprint can be very helpful in creating an even better, more coherent plan for the next sprint throughout all teams participating in the building of the product, however, it is at the Developers' disposal to decide if it is needed.

Also, similarly to the backlog refinement, it is possible to have representatives from other teams join the Sprint Planning of another team (limited to 1-2 Developers at most), as long as the meetings do not occur at the same time. The need for this should be identified by the team, due to any outstanding dependencies with another team or high-relevance Product Backlog items being undertaken by the other team.



6.2.3 Synchronization of daily work

Apart from obvious cross-team collaboration on a daily basis, there are 2 standardized approaches to fostering work-level synchronization between teams throughout the sprint. The first is an extension to the standard *Daily Scrum*, which is held by all teams on a regular, daily basis and the second is an additional meeting, called the *Scrum of Scrums*.

6.2.3.1 Daily Scrum

When working with multiple teams on a single Product Backlog it is recommended for designated participants of the teams (self-managing teams choose the appropriate member, which can change from time to time) to participate mutually in each other's Daily Scrum events. Having done that, the "exchange" participant shares his/her findings with the rest of his/her team; it allows teams to stay aware of each other's progress and issues, thus making it possible to synchronize daily work and address cross-team dependencies regularly. The time effort spent here is negligible, yet the benefit is huge.

While this is not obligatory, it is highly recommended for Scrum Teams, who especially have not established a good synchronization relationship with other teams working on the same product. This should be done in addition to the *Scrum of Scrums*, which is mandatory when working with multiple teams.

6.2.3.2 Scrum of Scrums (SoS)

This is a **30-minute per week** meeting, where at least 2 members of Developers (or more if considered required by the Developers), the Product Owners and Scrum Masters of those teams working on a single product meet. The meeting optimally takes place at a regular time and location. The meeting takes places after the Daily Scrum on the given day, in order to allow teams discuss their Sprint work on a team level first.

The following topics are discussed by the teams during the meeting:

- what work, that affects other teams, has been completed since the last SoS meeting?
- what work, that affects other teams, is planned to be done before the next SoS meeting?
- what key issues are being faced or addressed by the team which have impact on the common Product Increment?
- what dependencies between the teams exist and how are they being handled?
- is all work up to this point successfully integrated into the Product Increment?

If Scrum of Scrums level impediments are identified during the meetings, the Scrum Masters discuss who and how will take ownership in addressing those impediments.

6.2.4 Sprint Review

In a multi-team setup, the Sprint Review event takes place once, together with all participating teams and stakeholders. The Sprint Review remains a product-centric event focusing on the finished sprint and the delivered potentially releasable Product Increment (a single increment for all participating Scrum Teams).

In addition to the standard event, several team-building and collaborative techniques can be used to facilitate the large group, including Open Space, Review Fair, etc. (more in [8.3 - Examples \(see page 427\)](#))

The standard Sprint Review is described here: [4.5 - The Sprint Review \(see page 122\)](#)

A few things to consider during a cross-team, joint Sprint Review, per each of its distinct stages:

- summarizing the sprint - the Sprint summary is conducted by the Product Owners as a combined effort; the focus should always be on the overall product and while different Product Owners can contribute to their “piece of the puzzle”, the resulting conclusion must underline the status of the Sprint Goal for the product as delivered by all of the teams
- presenting a potentially releasable Product Increment - the Demo is conducted by the Developers responsible for delivering the particular functionality. It is good practice for the teams to Demo one after another, showing off their input to the Product Increment. In case of a very large increment or large number of teams, a *Review Fair* can be conducted in order to limit the amount of time needed for the whole review. That way the Demo is parallelized and interested stakeholders can choose which part of the Product Increment they would like to inspect in more detail.
- discussing and gathering feedback - the discussion during the Sprint Review involves all participants from all teams. When having 2 or 3 teams it should be still possible to conduct the discussion in a joint and orderly manner. In case the number of participants is very large, the discussion can be conducted in the form of an *Open Space* introducing the “law of 2 feet”. That way the discussion is segmented and parallelized, allowing all participants to be involved in the topics most relevant to them.
- making decisions and planning next steps - similarly to the summary, the final stage of the Sprint Review is held in a joint gathering, with the Product Owners and Product Manager playing the main

role. All other participants collaborate to provide feedback to the roadmap and support the decision-making process.

6.2.5 Sprint Retrospective

When considering the Sprint Retrospective for multiple teams working together in Sprints on delivering a single Product Increment, there is a bit more to be done in order to get the maximum value out of the event.

Two parts can be distinguished for a multi-team Sprint Retrospective:

1. A standard, single-team Sprint Retrospective conducted per each team individually, as described in: [4.6 - The Sprint Retrospective \(see page 126\)](#) (the events may run in parallel to each other, as long as the Product Owner and Scrum Master are separate for both teams)
2. A joint, cross-team, follow-up Sprint Retrospective

The joint Sprint Retrospective takes place only after all teams participating in work on a single product, have completed their team retros. The joint Sprint Retrospective is a **1 hour timeboxed event** taking place every sprint in a regular location and time.

The participants include (from each team):

- Developer representatives (1-2 team members at most)
- Product Owners
- Scrum Masters

The joint Sprint Retrospective is where the output of the team retros is further discussed on a product level, thus concentrating on general actions points, which may turn out to be global for all teams.

The steps for conducting this meeting are as follows:

- participants present findings from their team retrospectives
- participants discuss and collaborate to identify common issues on product level
- participants address identified issues, elaborating on action items, next steps and assignees
- participants share positive findings and success stories from their team-level retrospectives
- if time allows and/or if there are no common product-level issues identified, participants discuss significant team-level issues and collaborate to support each other in finding potential resolutions

Similarly to team-level retrospectives, the outcome of a joint retrospective is a limited list of action points to be improved on, being a part of the next Sprint Backlog. The action point can be covered with one of the team-level Sprint Retrospective action points, with the difference that it is being addressed with the context of the entire product, not just the team. Ownership of the items is to be agreed by the participants during the joint Sprint Retrospective.

6.2.6 Collaboration between teams with different setups

No matter the setup that the teams are working under, the collaboration between the teams need to cover the following aspects:

- synchronization of the upcoming scope of work
- synchronization in the run to cover all the touchpoints
- if needed - common presentation of the produced outputs and feedback gathering

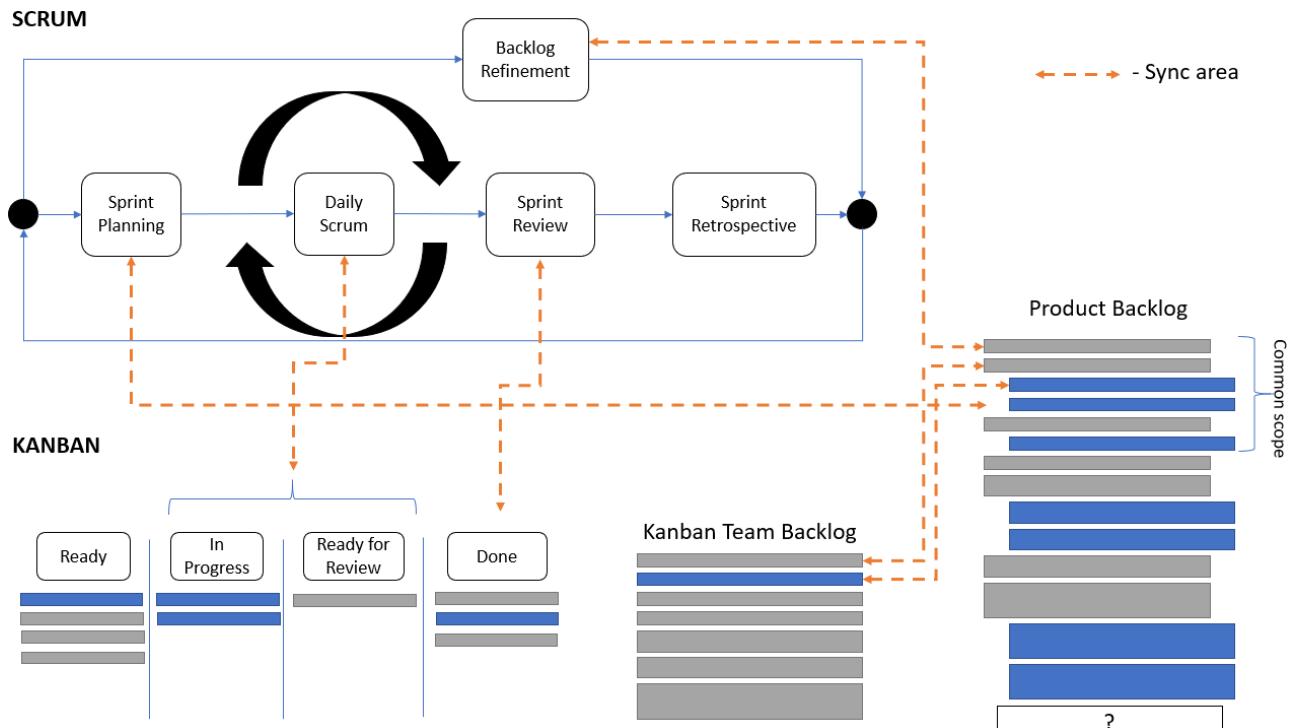
All the above is applicable when working on different variations of Scrum (Small Scale Scrum, consultancy teams, high-product-granularity teams). In the consultancy teams setup, we need to consider if there is a need to synchronize work or calibrate the activities with the consultants that are not part of the Agile team to make sure that we are having a proper information flow between the engaged parties.

Although the above three points are valid, the collaboration between **Scrum and Kanban** teams may look a bit different. Between these setups, we don't have common events to address the touchpoints so every time it occurs we need to establish a dedicated approach to synchronize the work. It can be done in multiple ways:

- establishing a dedicated common synchronization events
- using the above described Scrum events and inviting the Kanban teams representatives for synchronization
- Make sure that the synchronization takes place without establishing dedicated meetings or events

When planning the work that needs to be synchronized we need to make sure that the common scope of work is properly aligned between the teams. It is being done by properly planning upcoming sprints scope for Scrum teams and reflecting that in the proper backlog prioritization on the Kanban team/s side.

The below graphics is presenting the areas that need to be synchronized between Scrum and Kanban teams:



6.3 6.3 - Managing a single product backlog across multiple teams

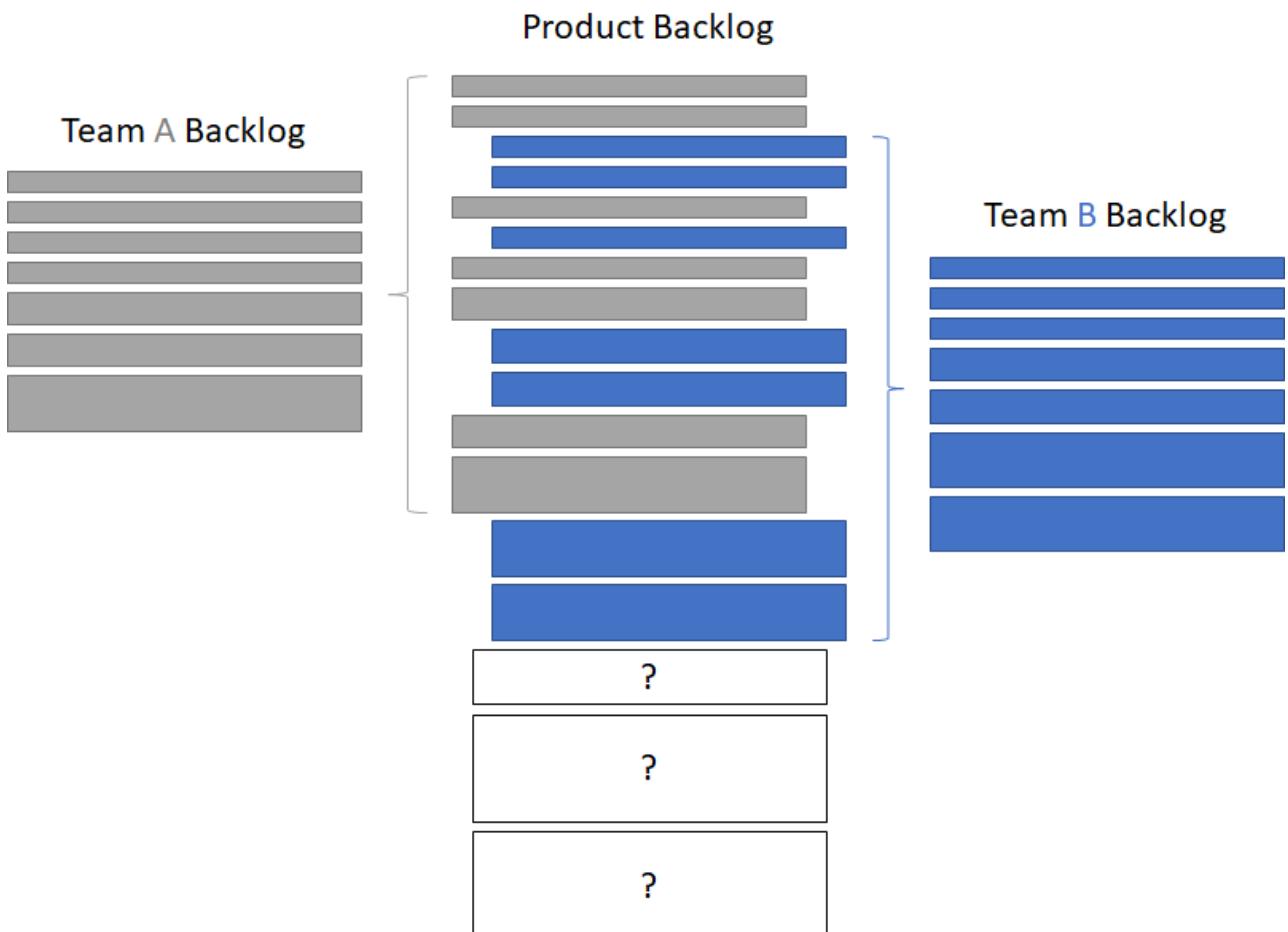
In the chapter dedicated to the synchronization of work across multiple teams, who are working a single, complex product ([4 - Events and working together in sprints \(see page 107\)](#)), it is described how to make use of the available Agile frameworks in order to make the agile delivery process feasible for all teams with respect to the spirit of Agile. Multiple teams who work to deliver a single, integrated Product Increment face some challenges in the various stages of the process, such as:

- synchronization and parallelization of the events
- coordination of work on a daily basis
- Product Backlog dependency management
- competence sharing and exchange of knowledge

One of the vital parts of having a successful, multi-team product setup is proper management of the Product Backlog for all the participating teams. Remember that one product has equally only one associated Product Backlog, where all the known/identified items required to complete the product are located.

6.3.1 Preparing team backlogs

Having multiple teams requires a certain level of collaboration, however, the teams also need to operate as independent organisms, which are capable of fully contributing to building the value of the Product. This facilitates the parallelization of work being done by the teams and introduces the right balance between a Product-focused collaboration and team uniqueness.



Preparing Team Backlogs is an ongoing process, similar to backlog refinement. The goal of this process is to establish two (or more, depending on the number of teams) relatively independent (to some extent only, as the items are still a part of the same Product Backlog) Team Backlogs which represent the upcoming work of independent Scrum Teams or Kanban teams. The activity, called *Pre-Planning* or *Cross-Team Pre-Planning* ([6.2 - Cross-team collaboration in a single product \(see page 219\)](#)) should be conducted by representatives of

the Developers and the Product Owners of the Scrum Teams, which are contributing to the growth of the single Product.

The *Pre-Planning* also works as a load-balancing and prioritization mechanism, which allows sharing the work across teams to make the most efficient use of their capacity. There are a few things that should be considered when taking into account the distribution:

- **domain knowledge specialization** - sometimes (however it is not mandatory) the teams can be specialized in specific domains of the Product; they can be then identified as “feature teams”. Should that be an attribute that has an impact on the distribution of the Product Backlog items, then it should be taken into account.

For example: one team specializes in the logistics module part of the system, while another is more oriented on the commerce module part of the system.

Despite the specializations, all teams should maintain 40-60% of their competences to be available to handle the entire Product Backlog.

- i** Team specializations should not impede the work of an Agile Team by limiting it to a certain domain, but rather they should allow the team to be a better expert on that domain than other teams.

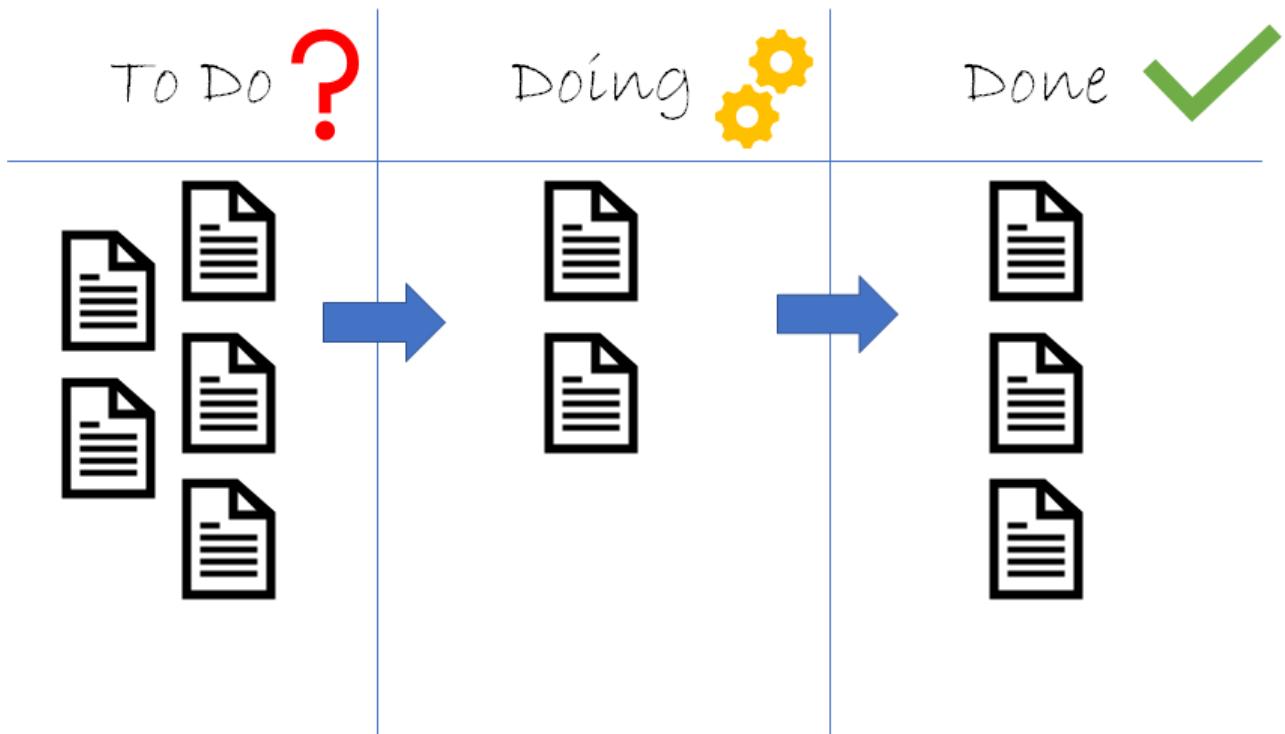
- **domain knowledge exchange** - if the expected result and goal, that we are trying to accomplish is spreading the team’s knowledge and experience, and perhaps sharing the knowledge about particular domains among other teams than the distribution of the items could be considered in a reverse pattern than that in the previous point. Assigning unfamiliar items (by a small percentage, such as 10%) to Scrum Teams, gradually raises their level of that knowledge in a stable, low-risk environment. During that time, the team can learn and acquire support from the other specialized team when necessary - one of the benefits of scaling multiple teams on a single Product Backlog.
- **backlog items priority** - when considering the distribution of Product Backlog items among teams, the priority with regard to the single Product Backlog should be well balanced. Try to avoid situations, where multiple highest priority items are assigned to one team. This gives a higher chance of completing the most valuable items by the Scrum Teams.
- **backlog items granularity** - when considering the distribution of Product Backlog items among teams, also do consider the size of the item at hand. Ensuring a balanced team backlog requires a varying dimension of items which are assigned to each team. If the Product Backlog contains various sizes of items, ensure that the team backlogs have this proportion maintained. Loading a single team with large items (assuming that they cannot be further broken down) will put a higher risk on the sprints of that team.

Proper grooming of the Product Backlog and pre-planning sessions with the affected teams, should allow building well-refined Team Backlogs, which in turn will make it easier to conduct seamless backlog refinements by the Scrum Teams in parallel. The teams will be able to create a plan not only for the upcoming cycle but a rough idea of their work agenda for 2-3 Sprints upfront, making it more transparent in terms of meeting the roadmap milestones.

6.3.2 Team Scrum boards

A Scrum board (for example in Jira) is used to facilitate and illustrate in a transparent way, the progress of a Scrum Team towards reaching their Sprint Goal. A Scrum board makes it easier to identify on a detailed level the tasks that need to be executed to complete the Sprint Backlog items (for example during Sprint Planning) and make it easier to collaborate, especially for distributed Scrum Teams.

Although there exists one Scrum board representing every single Product Backlog as a whole, it is highly recommended to maintain specific Scrum boards which help to focus from a given perspective. One such perspective is the Developers.



A team Scrum board allows having a view on:

- the Team Backlog, which represents items only dedicated for that particular team
- the Team's Sprint(s), which hold the current and future Sprint Backlogs for that particular team

While the teams do not miss the overview, by having the main Product Backlog available, they can focus in more detail on the scope of their most near work to be done.

Team Scrum boards are based on team issue filters, that indicate the list of items marked for a particular team during the *Cross-team Pre-Planning*. In Jira, issues are distinguished by the "Team name" field, where the name of the particular team is specified and should be selected.

6.3.3 Team Kanban boards

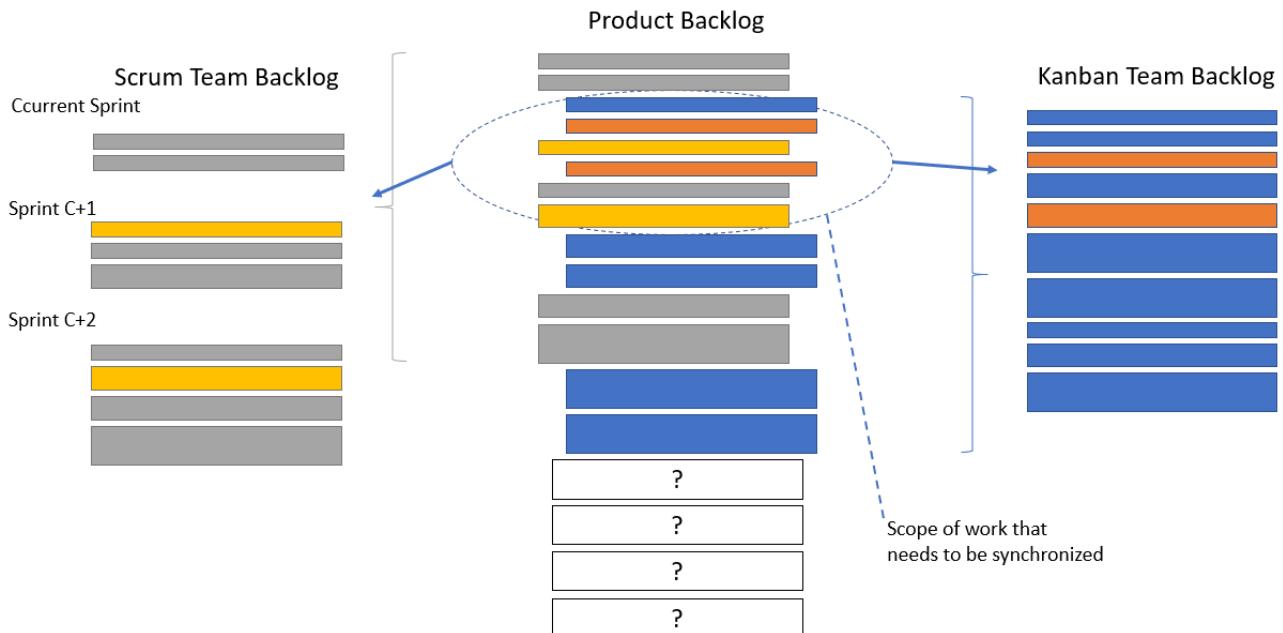
A Kanban team board is used to visualize in a transparent way the current state of the flow of work:

- the prioritized list of the scope of work waiting for realization on one side as well as

- issues currently being passed through the particular stages of the work process towards completion on the other one.

Identically as in Scrum team boards, Kanban Team boards are based on team issue filters, that indicate the list of items marked for a particular team.

Different from Scrum in Kanban, we are not closing the work scope in the specifically timed loops (Sprints), but we are managing the priorities and the current flow of work instead. These differences should be taken into consideration when managing the Product Backlog from a PO perspective. In Scrum, we can more or less plan in which Sprint the particular scope of work can be addressed. In Kanban to synchronize the work with Scrum teams, we have to plan the priorities in such a way (based on previous cycle time) that the scope of work will be synchronized across the teams. Please look at the below graphical presentation:



6.3.4 Managing the dependencies across multiple teams

The first rule of dependencies management is to eliminate or mitigate the dependencies first. This can be done in multiple ways:

- decomposing the issues into small enough pieces of work under which we don't have the dependencies;
- keeping the dependent pieces of work under one team, so we save energy that we would have to dedicate to the cross-team dependency management
- proper dependency visualization in Jira (through "linked issues" functionality)
- creation of the mid-term plans (i.e. using PI/Big Room Planning technique - for details please refer [HERE⁴³](#))
- by using the dependency management techniques, i.e. described below Dependency matrix or ROAM technique.

⁴³ <https://www.scaledagileframework.com/pi-planning/>

In general, it doesn't matter under which setup the team is working (EG Scrum, Kanban, Small Scale Scrum, consultancy teams, or high-product-granularity teams), the dependency management looks similar.

6.3.5 Dependency matrix

One of the good practices in building a comprehensive Product Backlog is avoiding dependencies, both external to the Product Backlog items and internal - between 2 or more independent Product Backlog items (epics, user stories, tasks, etc.). Nevertheless, no matter how well you plan out the backlog, sometimes dependencies cannot be avoided - it's just one of the challenges of building complex IT products. Those times, rather than trying to avoid dependencies, try to manage them. There's nothing more transparent and risk-reducing for a product than to be aware of dependencies and other aspects impacting the delivery of a final, potentially releasable Product Increment.

One of the ways to keep an eye on all of the dependencies in the Product Backlog is to create a *Dependency Matrix*. It is also valuable to keep those dependencies up to date by using proper links in Jira. Some tools, even allow us to build a dependency map or matrix based on those links or issue relationships. However, they are not critical to the process.

	User story 1	User story 2	User story 3	User story 4
User story A	-	low	high	-
User story B	medium	medium	-	-
User story C	high	-	-	-
User story D	low	high	-	high

Keeping a simple table of dependencies in the Confluence documentation space is surely an asset to the project. The table can be expanded and boosted with more details as necessary - it can be made custom to the needs.

The need for a transparent dependency matrix rises greatly when there is more than one team working on the Product Backlog. As you can imagine, it is much simpler to manage the dependencies where the entire scope is under the control of one team; that becomes more difficult with the need to synchronize across 2 or 3 teams.

6.3.5.1 Identifying dependencies

The process for identifying dependencies begins already from the very beginning of the user story's existence:

- the Product Owner can identify a dependency on a high-business level when creating the user story, when breaking down an epic, when working with the backlog in general
- the entire Scrum Team can identify a dependency during and in between backlog refinement sessions, by reviewing the Product Backlog item, by conducting feasibility studies
- the Developers can identify a dependency during the implementation of a feature, during conducting research work on a Spike backlog item, during some PoC - dependencies can be uncovered affecting issues which have not yet been developed

Always the person who has identified a dependency is responsible for reporting it properly into the matrix table and labeling all needed links in Jira. This way nothing gets lost - yes it requires a bit of time but saves so much more of it in the future. The dependencies should be discussed periodically during backlog refinements or at least when the affected or the affecting item is being discussed.

6.3.5.2 The ROAM technique

One of the techniques which facilitates dependency (risk and other issues as well) management is the *ROAM* technique. ROAM is an acronym, which defines 4 categories to mark dependencies:

- **Resolved** - indicates a risk or dependency which has been resolved, either already by having addressed the issue or having a detailed mean to address it in the near future; this indicates that the given dependency does not pose a risk for the development of the Product
- **Owned** - indicates a risk or dependency which is not yet resolved, but the risk associated with it and the outcome of the dependency is managed by a specified person, who is accountable for it
- **Accepted** - indicates a risk or dependency that has been accepted, since there is no known way to resolve or address it at the given time; usually, this impacts the work to be done by the teams, to consider proper communication and prioritization in order to avoid negative side effects
- **Mitigated** - indicates a risk or dependency that has been avoided by other means of addressing the issue, such as: changes in the Product Backlog or item description

By following this technique, managed dependencies can be categorized to dedicate the most focus to those which seem to pose the biggest threat and those which do not allow for an easy and quick resolution.

6.4 6.4 - Cross-product collaboration by one or more teams

When delivering multiple products we can distinguish two scenarios:

- a single team is delivering multiple products (high-product-granularity teams);
- multiple teams are delivering multiple products (within one Business Unit or across BU's or divisions)

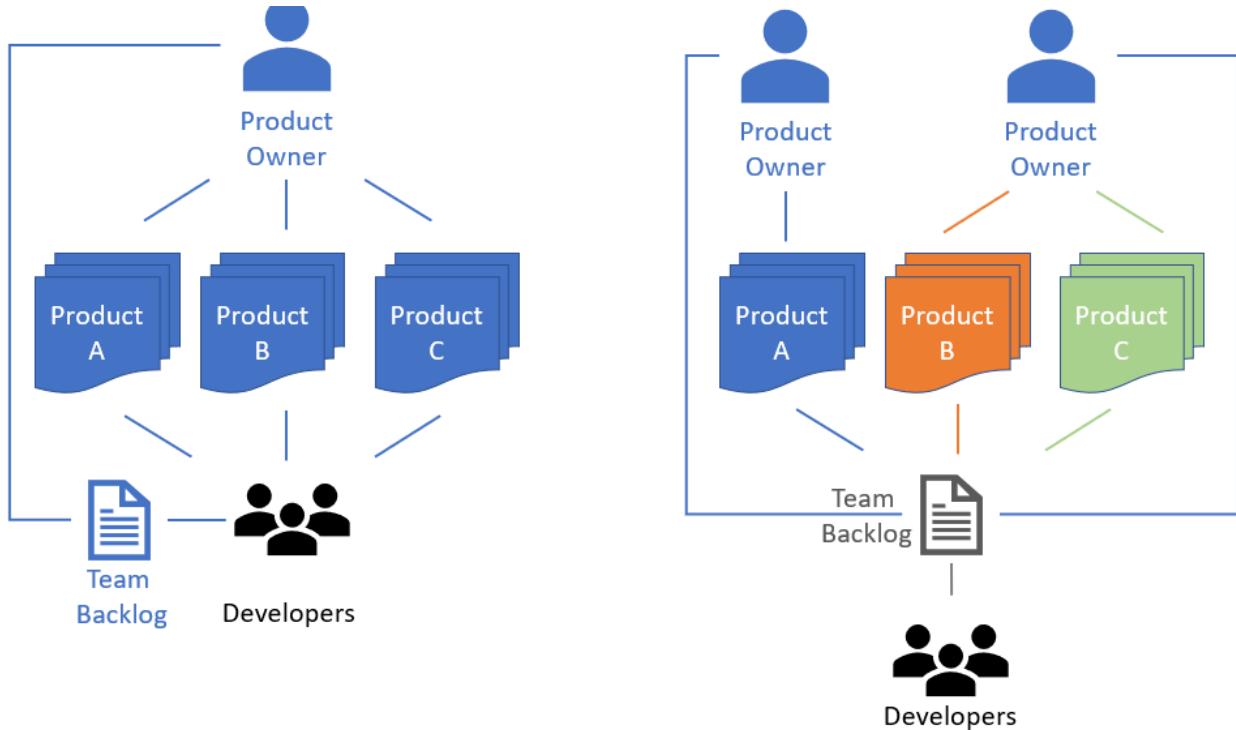
Each of the above setups requires a bit different approach, which you can read below.

6.4.1 One team working with multiple products

When a single team is delivering multiple products we can distinguish two variants of the collaboration:

- the team is collaborating with the single Product Owner who is covering all the products or
- the team is collaborating with multiple Product Owners covering different products

The distinction is visualized below graphics:



One team delivering multiple products with a single Product Owner is the most usual situation. In such a situation we have to remember the following aspects that are specific for this setup:

- **Deep vertical specialization will be an important factor for the scope of work planning.** In the high-product-granularity teams, it is natural that particular team members are covering specific products or product areas. This means that although PO can prioritize only one product development for the upcoming sprint/loop it won't be efficient to put a whole team to move forward with a single product. It needs a lot of dialog between the Product Owner and developers to find the most efficient balance that can address the product goals.
- **Prioritization in the longer perspective is crucial.** Keeping the above in mind If we want to put more emphasis on the specific products we need to plan them in the longer perspective. This may give the developers time to improve the knowledge shared in the specific areas so when the time comes, more team members can focus on these products. The more knowledge share we invest in advance, the more fast and efficient the delivery will be, reaching the expected goals.
- **Keeping the focus is always better than context switching.** Context switching always costs extra energy and time (please review [THIS⁴⁴](#) resource for more information). This means that the more focused we are throughout the Sprint or in the given time slot, the better results we achieve. From the

⁴⁴ <https://blog.rescuetime.com/context-switching/>

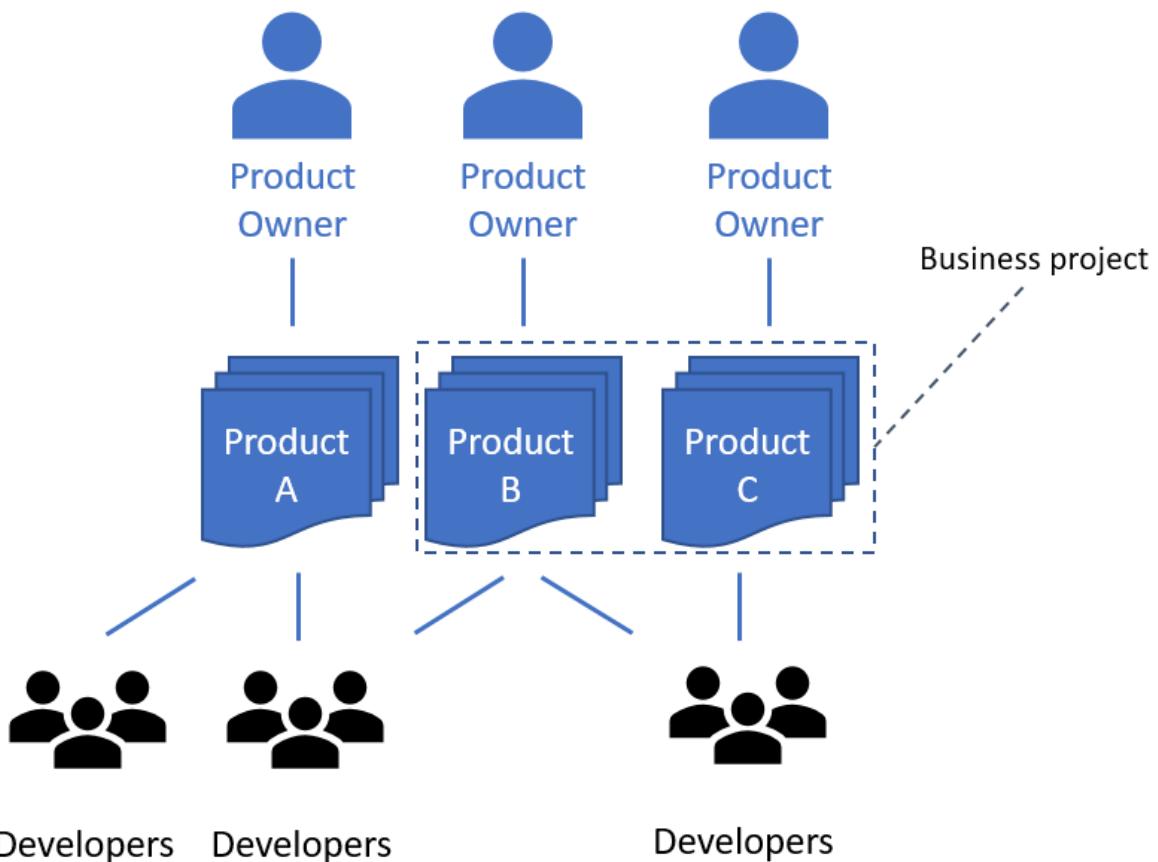
Product Owners perspective, it is better to have a single focus point in one sprint and switch it to another one than to split the attention between both in a single Sprint. This needs a bit more planning in advance, but will for sure benefit with a faster delivery rate.

- **The refinement process can be divided into multiple smaller meetings to improve efficiency.** More about this practice you can read here: [4.2 - The Backlog Refinement \(see page 113\)](#).
- **Whole-team estimation can be difficult.** Due to the fact that the knowledge about different products or their parts is not equally spread across the team, it may be very difficult to estimate the issues on the whole team level. In such cases, if there is no justification for common estimation, we can be based on the specific person's knowledge and judgment. Anyway, it is always good to have some dialog/discussion about the issues to improve the knowledge share process.
- **Everything goes to a single Team backlog.** No matter how many products the team is supporting, everything should be grouped in a single Team backlog where the final prioritization from the whole scope perspective is being made to avoid any misunderstandings.

One team delivering multiple products with many Product Owners. This setup is more challenging. All the above-stated points are valid here, but we are having an additional layer of coordination – between Product Owners. There is no additional event for Product Owners calibration which means that the calibration and coordination should be done always when needed. PO's should especially align the priorities between each other so we avoid some hot discussion or priorities shuffling with the Developers. If any conflicts of interest occur between PO's, should be decided by the Product Manager. We should especially remember here about context switching reduction and proper Sprint Goals crafting (in Kanban proper prioritization of the backlog). It is also good to discuss during the Refinement if there is anything that we could re-use from other Products supported by the team, so we may achieve some synergy effect.

6.4.2 Multiple teams supporting multiple products delivery

The most challenging setup occurs where multiple teams are supporting multiple Product delivery. Such setup example is shown on the below graphics:



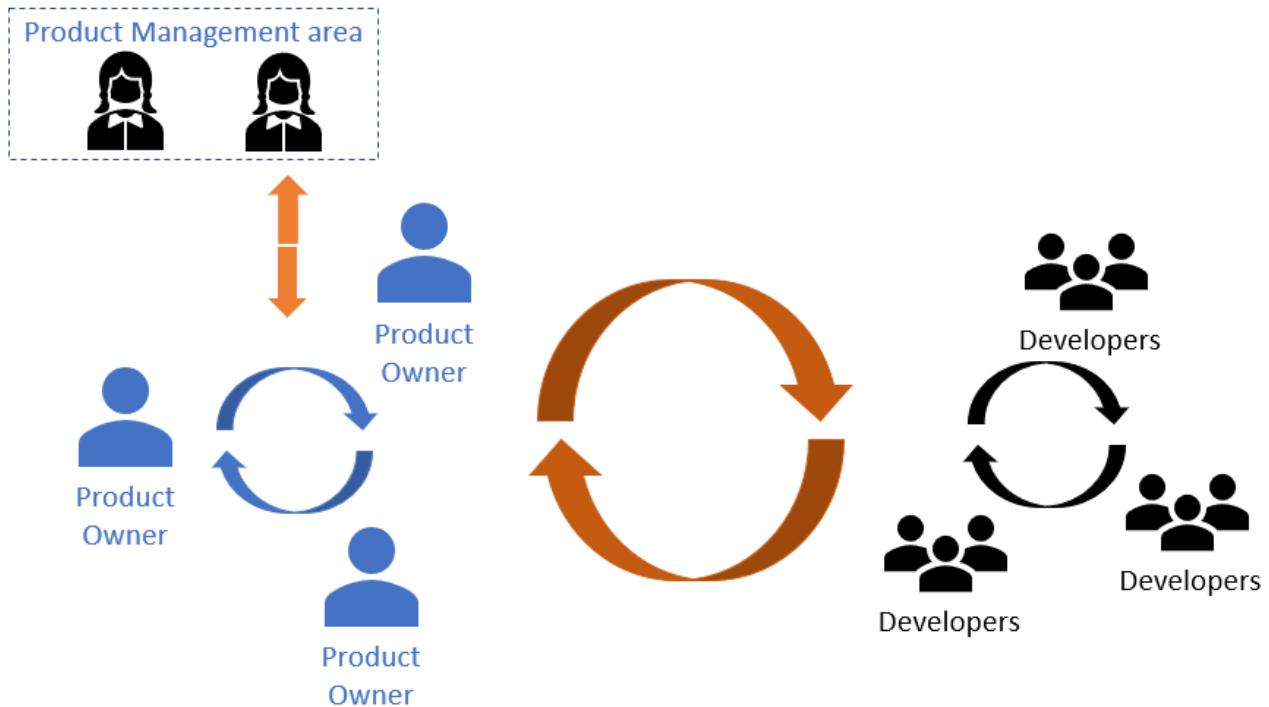
Having such a setup you should **consider the following suggestions:**

- Most of the Cross-team collaboration and dependencies management described in [6.3 - Managing a single product backlog across multiple teams \(see page 225\)](#) chapter are applicable to multiple teams supporting multiple products context.
- To reduce the complexity of such a setup it is good to split the Product coverage in a way, that there are as few common areas as possible (following the first rule of the scaling - "do not scale"). When they occur the Product Owners (or Product Manager/s) should decide who will take the interfering areas. When any sort of conflict regarding the priorities or Product handling occurs, the final decision about it always belongs to the Product Manager.
- The Product Owners should collaborate with each other and with the Product Manager on the area of Product vision, risk management, and the middle to long-term Product development plans.
- Having the common events from multiple teams and Products perspective may be challenging, for such events it is good to have a well-crafted agenda that will structurize the meetings and events. Having many Scrum Masters available it can be agreed that the facilitation of the events and meetings can rotate to have the full engagements from each side perspective
- A crucial event for that calibration is a cross-team pre-planning event. When having many teams and many Product Owners the event can be split into two parts to improve the event efficiency. The first part should be handled together to cover all the priorities, dependencies, and calibration on the common level. When done, each Product Owner can gather the team responsible for his/her scope

for detailed refinement. If one Product Owner is covering multiple teams, the teams can be handled sequentially.

- To make sure that all the needed information and documentation is properly stored and accessible by everybody engaged in this setup, common Confluence spaces or pages can be created and access properly granted. If needed, the setup can be consulted with an Agile Coach or DevOps Engineer from the NGA.

The below graphics is showing all the calibration that need's to be performed on multiple levels.



6.4.2.1 Jira overview

There are two ways of distinguishing the common areas from different Projects (products) in Jira:

- Jira business projects or
- Jira dedicated boards, that will cover the scope from multiple Jira projects. You can read more about the board setup here: [7.1.11 - Jira common boards setup for cross-product collaboration \(see page 306\)](#)

The scope of each team's work will be always visible under the Team board, as soon as we properly select the team name field in Jira.

Each above-described way is a bit different but can give you an overview of the common work areas across different projects and teams.

6.5 6.5 - Managing priorities, dependencies and roadmaps cross-product

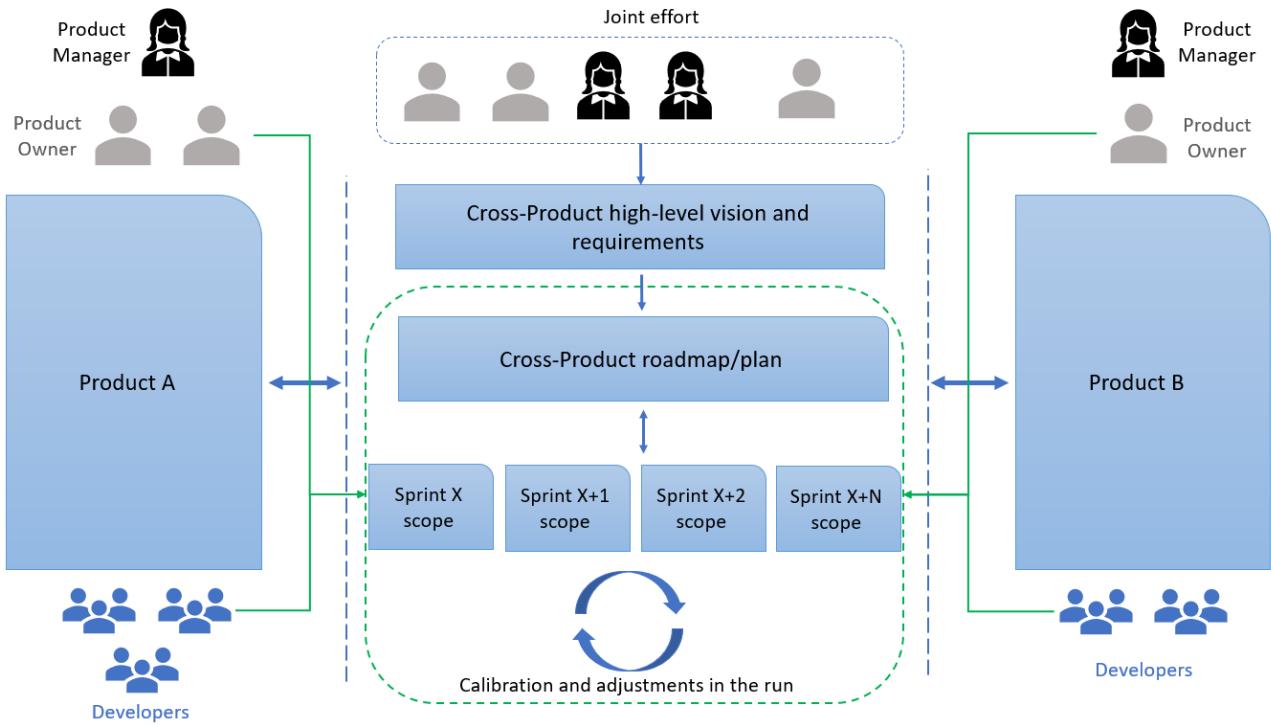
Cross-product planning and alignment of the work between many engaged persons and teams may be challenging. Such endeavor needs a lot of initial work being done between Product Managers and Product Owners and many adjustments in the run between all engaged parties. In this chapter, you will get hints and suggestions on how to do it in an organized manner not to get lost in the whole complexity.

6.5.1 Cross-product vision, roadmaps/plans, and scope creation

In some situations, it is justified to join the forces across products or business units even if this is a complex task. There may be multiple reasons to do so:

- **cost and time savings** - it is obvious that producing some features or system parts that can be reused in multiple products is generating substantial cost savings. The same goes for the time and energy that we need to dedicate in each product to achieve the expected results. In such a case, we can dedicate the saved time and energy to develop other Product features, so we gain a bigger product increment at the same time, delivering more value to the customers faster.
- **fast time-to-market** - in some cases, especially if we are operating in a highly competitive market, being faster with highly expected solutions, can give us a substantial competitive advantage. This can give us a bigger market share or higher gains.
- **market risks mitigation** - if we are going to introduce solutions that might be risky and a failure rate is high, mitigating such risk by spreading the investments across multiple products can be a good strategy.
- **proof of concept checkup** - cross-product joint effort may be a good idea if we want to check some concepts at the market fast.
- **sharing components** - utilization of shared services/components/other artifacts across teams and products can lead to a significant acceleration of progress and limitation of delivery costs, thanks to reuse (more: [Shared Components⁴⁵](#))
- **technical debt handling** - if across the products we are having similar technical debt in the same technology, handling it together can work.
- **joining the competence and skills not present in a single product** - joining the effort across products can give us good results in the situations where we don't have enough resources or competence under a single product

⁴⁵ <https://confluence.eg.dk/display/NG/Shared+Components>



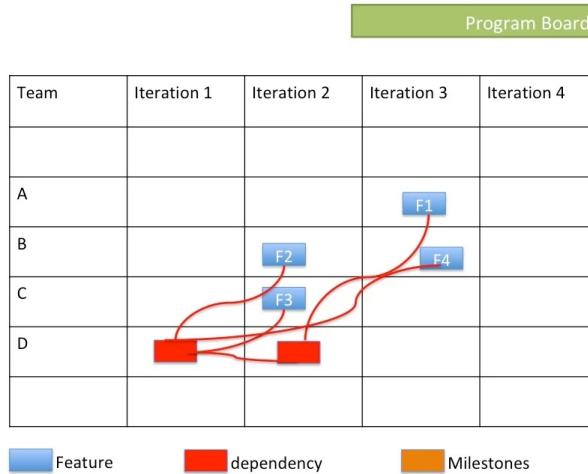
6.5.2 Big Room Planning

Big Room Planning is one of the popular techniques enabling to have a helicopter view on a program, consisting of many products and requires many development teams involvement. The purpose of an event is to have a high-level plan for the next 3 months including timeline and dependencies between all involved teams. It's a two-day event of planning together with all program and team members. Based on the master plan and program goals, all teams are talking through how they each will contribute to reach the program goals during next three months. Usually, they do that in a structured way by breaking down the epics from the master plan into features, with consideration of "the big why", the purpose of a program as a whole. Equally important, sharing "the smaller why", which is the goal for the next three months.

6.5.2.1 Preparation for Big Room Planning.

- Master plan creation. It's important input required for BPR. The master plan sets the direction for the entire program and helps all teams align. It should present a business context and vision of a program. Its the Program Manager and key stakeholders responsibility.
- Agree on who will share and update everyone on the program vision.
- Agree on who will demonstrate the current state of the solution and the overall architecture.
- Talk to the program stakeholders, and make sure that you get the right people in the room. You want to have sufficient knowledge and mandate present at your big room planning.
- Have all the epics of the master plan – with estimates and priorities
- Arrange facilitators according to the maturity level of the team. The lower the maturity level, the more facilitators you need. Up to one facilitator for each team, if they are green on Agile, breaking down epics and estimating them using planning poker.

- In case of an onsite event, prepare decks of planning poker, post-its, markers, red string for dependencies, tags, physical program board (including team names and sprints in one canvas)



- Get the right room and set it up. Create virtual setup accordingly for an online event.
 - Make sure to get a room that is big enough for everyone – and to hold one table for each team plus one table for the other program roles and stakeholders. One room, not two or more rooms close to each other, because the flow of communication between teams and people will suffer if you are not all in the same room.
 - Set up the room – preferably the day before the big room planning. Get the tables organized. Hang up the master plan where everyone can see it. Get the team-planning posters up on the walls by the teams' tables. Get the program board ready, and display relevant program posters – e.g., the program vision, the use case diagram, etc.
 - Show the start and end dates for each sprint on the program board, to make it extra clear that this has been decided upfront, and to make it easier for everyone.
 - Get the teams to write the names of the product owner and the Scrum master on the program board to make it easier for all participants. There's a lot to comprehend and remember; let's make these names not one of the things people have to remember.
 - Have name tags for all participants, to make it easier for people to remember each other's names which also helps build relationships.
 - Prepare agenda for the event (see example in execution chapter)
 - It's good to have a final briefing with your fellow facilitators and key stakeholders, the program leader, a program architecture responsible person, and business stakeholders (those representing the employees and/or customers, who will benefit from the product developed by the program), to run through the program and the roles one final time.

6.5.2.2 Execution of Big Room Planning.

Big Room Planning execution goes according to previously created agenda. Here is an example:

Agenda item	Remarks	Presented/facilitated by
Day 1		
Purpose	We're here to make a plan for the next 3 months, where the program wants to achieve XYZ.	Program leader
Program vision	Present/remind people about the (updated?) program vision	Business stakeholder
Solution & Architecture	How far are we with the solution, and what is the (updated?) foundation we're building the solution on?	Program architecture responsible
Master plan	Present the master plan with a special focus on the next 3 months	Facilitator (possibly = program leader)
Teams & epics	Which teams have stakes in which epics	Facilitator
Team breakout intro	Explaining the specifics of the teams breaking down epics into features, and estimating and prioritizing them	Facilitator
Team breakout	People are working in their teams	Facilitators
Program board	Teams post their initial plans on the program board and start coordinating with other teams	Facilitator
Repeat	Team breakout and program board discussions are repeated – typically 1-2 times before the final program board gathering	
Day 2		
Team breakout	Continued from day 1, while teams are encouraged to coordinate with other teams and stakeholders	Facilitators
Program board	Finalizing the program board with all teams features and dependencies	Facilitator

Risks	Teams are bringing up risks, and they are discussed and mitigated in the plenum	Program leader
Program objectives	Teams formulate their essential business objectives and then the program objectives for the 3 months are agreed upon.	Product owners and business stakeholders
Confidence vote	On a scale from 1-5 – how much does each team believe in their plans	Scrum masters and facilitator
Next steps	Usually, the next step is for the teams to make a sprint planning, and then get going on the work	Program leader
Keep & try	Reflecting over the 2 days of the big room planning	Facilitator

This agenda is inspired by the SAFe® PI Planning suggested agenda which is also a good reference point for event.

6.5.2.2.1 1. Purpose

Purpose of BRP is to craft a plan for the next three months. Start with sharing high-level program goals (XYZ).

If this is one of the first big room plannings in the program, it's good to make it clear that most things will probably go well, and that there will most likely also be some hiccups during the 2 days. People should stay open and treat it as learning opportunities – something to improve for next time.

The program goals are the helicopter view on what's in the master plan for the next three months, not just the individual epics, but also the purpose and a brief description, the essence, of what is planned. So this is reminding everyone of "the big why", which is the purpose of the program as a whole. And equally important, sharing "the smaller why", which is the goal for the next three months.

6.5.2.2.2 2. Program vision

Present/remind people about the (updated?) program vision

Here we want to get a key business stakeholder to give a pep talk, to remind us about why we're here. To explain why the program's strategic objectives are (still) important for the business and for the company as a whole. Storytelling is better than Powerpoint for this, and even better if we can get the stakeholders to tell their own story about "why this is important to me personally."

6.5.2.2.3 3. Solution & Architecture

Remind everyone how far we are with the solution, by giving a brief live demo of the actual product, or by having a visual overview of what has been delivered, and what is still missing. Also, repeat the architectural guidelines, and share the newest changes of the technical foundation.

6.5.2.2.4 4. Master plan

Have everyone get up, and “walk the master plan” together while explaining it. Spend most of the time on the next three months. Also, cover the following quarters, just to remind people about what we’re not working on now – but spend less time on that.

6.5.2.2.5 5. Teams and epics

Which teams have stakes in which epics. The goal of this part is to craft a map of epics that belongs to a specific team. Here is one example of how it could be done in practice:

- Epics selection. Each team should choose the epics they think belong to them, with little or no discussion with the other teams about it.
- Epic contribution mapping. The teams should gather around the epic overview board, and this time they should think about all the epics in which they played a role, even if it was a small role. The team representatives should put a small colored sticky note representing their involvement in those epics. After some time (~one hour), we should get an overview of which teams will be primary drivers on which epics – and which other teams were also involved in each epic.

In practice there is a lot of people standing by the board, coming and going throughout the two days, and it's usually people from different teams having a discussion about what they need from each other. How they could help each other?

6.5.2.2.6 6. Team breakout intro

This part is about explaining the specifics of the teams, breaking down epics into features, estimating and prioritizing them. Reminding people about the remaining agenda for the two days, and presenting all the details of the process. I usually include things like:

1. For each team – who will be your main facilitator
2. Have each team get an overview of the team’s capacity for each of the next five sprints. An example: you have a team with six people including the Scrum Master and the Product Owner. One person has one week’s vacation during the 2-week sprint. Each person counts 4 points per full week. So, on this team, the capacity for this sprint is $5 \times 8 + 1 \times 4 = 44$. Get the teams to write this on their team sprint posters and on the program board. A couple of comments:
 - a. Get all teams to and share their team capacities with the other teams.
 - b. History has proven, that people have always been too optimistic during planning. Counting in some buffer (~10-20%) is a mechanism to make up for the optimism.
 - c. Why include the Scrum master and the product owner in the capacity? Because building a product is far from only “putting it together,” e.g., writing and testing the code in software

- programs. It also includes a substantial amount of business discussions, decisions, coordinating etc. In other words, the work of the Scrum master and the product owner.
3. Next, explain how we want teams to break down their epics in 3-5 features per epic (with fewer features you're only scratching the surface – with more, it is impossible to get the overview at the program level).
 4. We want the features on Post-its. For each feature, ask for a short name, a description, a reference to the epic, and an estimate.
 5. When the estimation starts, start by finding a feature, that can be achieved by one person in one week or by two people in 2.5 days. Give it a 5, and estimate other features relative to that.
 6. Other than that, the teams can just go ahead and break down into features – just like they would break down into stories in a Scrum sprint planning session.
 7. Repeat that teams with regular intervals will be asked to share the results of their planning with the other teams by the program board.

6.5.2.2.7 7. Team breakout

Simply let people follow the instructions from the Team breakout intro. Make sure, that the teams get the right amount of attention from the stakeholders and facilitators. Not too much to disturb them, and also enough to give them the guidance they need.

6.5.2.2.8 8. Program board

Teams post their initial plans on the program board and start coordinating with other teams.

Give teams a 15 to 20-minute warning before the first gathering at the program board. Ask them to write a Post-it for each of their features, but for the program board without the description in order to create a cleaner and better overview.

When the time is up, get everyone up by the program board, and get one from each team (probably the Scrum master or the product owner) to place their features in the sprints where the team thinks it belongs.

You might need to encourage a discussion about dependencies between the teams and features, but chances are that the teams will start discussing this without your help when they have it in front of them.

Help visualize the dependencies by connecting features that are dependent on each other with a red string. Encourage teams to finish discussions about dependencies on the side once the dependencies have been identified, unless it is a dependency that involves all teams.

A tip: have the first gathering by the program board earlier rather than later- no later than a couple of hours after the first team breakout starts. You will probably get some resistance from the teams, because they do not feel ready yet. If you wait too long, you will not get the cross-team collaboration you get if you do it early. It is amazing to see how teams are cross-pollinating a lot more after the first program board gathering.

6.5.2.2.9 9. Repeat

Team breakout and program board discussions are repeated – typically 1-2 times before the final program board gathering. Remember to gently force them to meet by the program board every 1-2 hours – even when

they're not feeling ready in the individual teams. It will nudge them to collaborate and coordinate more across teams.

6.5.2.2.10 10. Team breakout

Continued from day 1, while teams are encouraged to coordinate with other teams and stakeholders

After this team breakout, you should start to see the outline of plans for each team. If they have not put their features up on the team planning posters yet, now is a good time to get them to do that. It makes it easier for the teams to understand what the other teams are planning, when they visit each other.

6.5.2.2.11 11. Program board

Finalizing the program board with all teams' features and dependencies. Remember that the plan just needs to be good enough. It will be revisited every day, and details will be sorted out later.

6.5.2.2.12 12. Risks

Teams are bringing up risks, and they are discussed and mitigated in the plenum. Some risks result in a mitigation activity, which we want to go on a team's plan and/or on the program board, so we do not just talk about it, but actually do something about it. Other risks are just accepted. Spend most of the time on the top 3-5 risks, and just list the others for now.

6.5.2.2.13 13. Program objectives

Teams formulate their essential business objectives and then the program objectives for the 3 months are agreed upon.

In a program with many teams, opinions, epics and features, people often need help to navigate and remember what the overall purpose is.

To help with the navigation teams need to be guided by the product owner, to describe a few overall goals for the three months. Then we ask the product owners to get together with the business stakeholder(s) to discuss and agree on a few program objectives for the three months. The fewer and shorter the objectives are, the stronger they are in order to help the entire program navigate towards a common goal.

Here's an example of a program objective from a program, that really struggled to go live: "Run on live data"

6.5.2.2.14 14. Confidence vote

On a scale from 1-5 – how much does each team believe in their plans

As a final test, we can get the Scrum masters for each team to ask the team members about how much they believe in their team's plan, and possibly also how much they believe in the plan for the entire program, by a show of fingers up to 5.

Suggestion: Do this only if you follow up with a discussion about why the numbers are not higher, and most importantly, how the plans or circumstances can be changed so that people can believe more in the plans. A good question is: "What would it take to get you closer to a 5?"

6.5.2.2.15 15. Next steps

Usually, the next step is for the teams to make a sprint planning, and then get going on the work.. Also, talk about which other things will happen in the near future, and remember to thank everyone for their time and corporation.

6.5.2.2.16 16. Keep & try

Reflecting over the two days of the big room planning

Remember the opening comment about potential hick-ups during the big room planning?

This is where you gather input for how to make it better next time. One example is asking people to write the following on a Post-it:

- A number from 1-5 (5 is best) for what they thought about the two days
- One thing to keep for next time
- One thing to try to do differently next time

Get people to put them on the door on the way out. Take a picture when everyone has left – maybe read them, but wait until you prepare your next big room planning before you think much more about it. At this point in time, after two days of big room planning, your brain is probably fried, not able to reflect rationally anyway.

7 7 - Jira and Confluence manuals

- [7.1 - Jira \(see page 245\)](#)
 - [7.1.1 - Estimating hours effort in Jira \(see page 246\)](#)
 - [7.1.2 - Planning and starting a Sprint \(see page 254\)](#)
 - [7.1.3 - Using the Active Sprint dashboard view \(see page 260\)](#)
 - [7.1.4 - Closing a sprint \(see page 266\)](#)
 - [7.1.5 - Using the KanBan board and extended backlog \(see page 268\)](#)
 - [7.1.6 - Jira issue types and hierarchy \(see page 272\)](#)
 - [7.1.7 - Jira issues workflow \(see page 279\)](#)
 - [7.1.8 - Personalized MS Outlook rule for Jira notifications \(see page 287\)](#)
 - [7.1.9 - Creating custom HTML links to create Jira issues \(see page 290\)](#)
 - [7.1.10 - Jira JQL hints, useful examples and managing filters \(see page 293\)](#)
 - [7.1.11 - Jira common boards setup for cross-product collaboration \(see page 306\)](#)
 - [7.1.12 - Jira Automation \(see page 314\)](#)
- [7.2 - Confluence \(see page 318\)](#)
 - [7.2.1 - Confluence space structure \(see page 319\)](#)
 - [7.2.2 - Documenting a Sprint Retrospective \(see page 321\)](#)
- [7.3 - Versioning and release notes \(see page 323\)](#)
 - [7.3.1 - Documenting Release Notes \(see page 332\)](#)
- [7.4 - Dashboard metrics description \(see page 339\)](#)
- [7.5 - Logging time and transferring it to NetSuite ERP \(see page 344\)](#)
- [7.6 - User management \(see page 355\)](#)
- [7.7 - Instruction to GDPR and threat assessment \(see page 364\)](#)
- [7.8 - Jira & Confluence permissions schema \(see page 373\)](#)
- [7.9 - ServiceNow integration \(see page 376\)](#)
 - [7.9.1 - Jira - ServiceNow integration board and issue flow \(see page 377\)](#)
 - [7.9.2 - Jira - ServiceNow integration fields mapping \(see page 380\)](#)
 - [7.9.3 - Jira - ServiceNow integration communication via comments \(see page 384\)](#)
- [7.10 - \[FAQ\] Cloud to On-Prem migration \(see page 386\)](#)
- [7.11 - Jira Advanced Roadmaps \(see page 396\)](#)
- [7.12 - EG Security default schema for GDPR data access \(see page 407\)](#)
- [7.13 - AI Metrics for software development \(see page 410\)](#)

7.1 7.1 - Jira

- [7.1.1 - Estimating hours effort in Jira \(see page 246\)](#)
- [7.1.2 - Planning and starting a Sprint \(see page 254\)](#)

- [7.1.3 - Using the Active Sprint dashboard view \(see page 260\)](#)
- [7.1.4 - Closing a sprint \(see page 266\)](#)
- [7.1.5 - Using the KanBan board and extended backlog \(see page 268\)](#)
- [7.1.6 - Jira issue types and hierarchy \(see page 272\)](#)
- [7.1.7 - Jira issues workflow \(see page 279\)](#)
- [7.1.8 - Personalized MS Outlook rule for Jira notifications \(see page 287\)](#)
- [7.1.9 - Creating custom HTML links to create Jira issues \(see page 290\)](#)
- [7.1.10 - Jira JQL hints, useful examples and managing filters \(see page 293\)](#)
- [7.1.11 - Jira common boards setup for cross-product collaboration \(see page 306\)](#)
- [7.1.12 - Jira Automation \(see page 314\)](#)

7.1.1 7.1.1 - Estimating hours effort in Jira

Most frequently Agile Teams estimate Product Backlog items using some form of abstract, relative estimation. One common technique is using Story Points and Planning Poker, for example. Nevertheless, due to various reasons (as consulted with the team manager) it may occur that teams estimate items using regular time-based metrics, such as hours, minutes, etc. While Story Points estimation in Jira is quite straightforward, using hourly based estimations on different issue type levels, may introduce some complexity.

Recapping the basics, as in [5.2 - Refining and estimating an initial Product Backlog \(see page 152\)](#) - estimations in EG Scrum are done for Product Backlog items (user stories, tasks, bugs, and sub-tasks) as early as during backlog refinement sessions but no later than during Sprint Planning. Depending on the estimation type selected in KanBan (frequently issue count is the preferred metric, which does not require estimation but rather a proper refinement and granularity) items are usually estimated as soon as during backlog refinement and no later than being marked ready for development. Of course, estimates assessed during backlog refinement should be evaluated and confirmed finally during Sprint Planning (EG Scrum) or before starting the development of the issues. Estimating epics is another topic, which is related to roadmap planning and high-level estimation, which is not in the scope of this manual.

7.1.1.1 Defining an estimate when creating a user story

It is possible to define an *Original Estimate* already when creating a user story. In order to do so, simply enter the value expressed in **hours (default)**, **minutes** or **days** into the field “Original Estimate”. Some possible values are:

- 6 - this will translate into 6 hours, as hours is the default metric (the equivalent of 6h)
- 30m - this will translate into 30 minutes
- 2d - this will translate into 2 days

Combinations are also possible, for example:

- 2h 30m - this will translate into 2 hours and 30 minutes
- 1d 4h - this will translate into 1 day and 4 hours

- (i)** Please note: the default duration of **1 day is set to 7 hours**, therefore whenever the number of hours declared exceeds 7 it will convert into days; entering a value such as 12h will result in the presentation of 1d 5h

Create Issue

Attachment [Configure Fields ▾](#)

Assignee [Assign to me](#)

Reporter* [Start typing to get a list of possible matches.](#)

Component/s **None**

Team name [EG Default Team name field](#)

Story Points [Measurement of complexity and/or size of a requirement.](#)

Sprint [Jira Software sprint field](#)

Labels [Begin typing to find and create labels or press down to select a suggested label.](#)

Original Estimate (eg. 3w 4d 12h) [?](#)
 The original estimate of how much work is involved in resolving this issue.

Remaining Estimate (eg. 3w 4d 12h) [?](#)
 An estimate of how much work remains until this issue will be resolved.

Issue category* [▼](#)

ERP Activity [AX Activity ID](#)

Epic Link [Choose an epic to assign this issue to.](#)

Create another

7.1.1.1.1 Remaining Estimate

Below the *Original Estimate* field, there is a *Remaining Estimate* field which indicates the amount of work remaining from the declared original value. Upon creation of a user story, this field when left empty will populate with the value from the Original Estimate field. In case of a rare scenario, should any work already have been completed when creating the user story, then it is possible to define how much time is left - if different then the Original Estimate.



Please note: the Remaining Estimate value is copied from the Original Estimate value only upon the first issue estimation (an estimate is entered into the Original Estimate field). During later work, only the Remaining Estimate field should be manipulated to reflect the actual remaining work. Should for any reason, the Original Estimate field be updated after the creation of the issue, the Remaining Estimate value needs to be adjusted manually.

7.1.1.2 Defining an estimate on an existing user story

If a user story is already created, then it is possible to define an estimate directly from the backlog view in Jira. In order to do that, simply select the user story which needs to be estimated (its details should appear on the right-sided pane), click on the *Estimate* field to go into edit mode, and enter the estimated value using the format described above.

For a new user story, which was not estimated previously it should look like this in **Backlog view**:

The screenshot shows the Jira Backlog view. On the left, a list of issues is visible, including "TRT-1 Test story 1" (6h), "TRT-2 Test story 2" (8h), "TRT-3 Test story 3" (4h), and "TRT-6 Test story 4". The fourth item, "TRT-6 Test story 4", is highlighted with a red box. On the right, the details for "Test story 4" are shown in a modal window. The "Estimate" field is empty, and the "Remaining" field is set to "Unestimated". A button labeled "Original Time Estimate" is present next to the Remaining field. The "Details" section shows the status as "TO DO", priority as "Medium", and other fields like Component/s, Labels, and Dates are listed.

Please notice that the *Remaining (time)* field shows “Unestimated”. Once the estimation has been entered and confirmed, it will be copied from the *Estimate* value and will look like this in **Backlog view**:

Backlog

The screenshot shows the Jira Backlog interface. On the left, there's a list of issues under 'Backlog' with 4 issues: TRT-1, TRT-2, TRT-3, and TRT-6. TRT-6 is highlighted with a red box. On the right, the full issue view for 'Test story 4' is displayed. The 'Estimate' field shows '5h' and the 'Remaining' field also shows '5h'. The 'Details' section includes fields like Status, Priority, Component/s, Labels, Affects Version/s, Fix Version/s, and Epic Link. The 'People' section shows reporter 'Dariusz Szlek' and assignee 'Unassigned'. The 'Dates' section shows creation and update times. The 'Description' section has a placeholder 'Click to add description'.

Estimating an existing issue is also possible in the **Full issue view**. In order to do so, it is necessary to select the Edit button and enter the *Original Estimate* value similarly as during the creation of a new issue and click the Update button:

The screenshot shows the 'Edit Issue : TRT-6' dialog box. The 'Original Estimate' field is highlighted with a red box and contains the value '(eg. 3w 4d 12h)'. The dialog also shows 'Remaining Estimate' with the same value. Other fields include Priority (Medium), Labels (None), Sprint (Jira Software sprint field), and TFS-ItemID. The right side of the dialog shows the issue details, people, dates, development, agile, and hipchat sections.

Once the issue has been estimated, the *Time tracking* module should appear in the full issue view (only) with estimation and time remaining / time logged details:

The screenshot shows a Jira issue details page for 'Test story 4'. The page has a header with project name and issue key, and a toolbar with edit, comment, assign, more, ready for development, start progress, and admin buttons. Below the toolbar are sections for 'Details', 'Description', 'Attachments', 'Activity', and 'Time Tracking'. The 'Time Tracking' section is highlighted with a red box and contains three bars: Estimated (blue, 5h), Remaining (orange, 5h), and Logged (grey, Not Specified).

To summarize, the issue time estimate is reflected in 3 main places:

- Backlog view
 - in the *Estimate* field, the input estimate is presented or displays “Unestimated” if empty
 - on the Backlog, the hourly estimate is presented on the Item inline
- Full issue view
 - in the *Time Tracking* module, the time statistics are presented (only if not empty, otherwise the module is not shown):
 - Estimated time (*Estimated*)
 - Remaining time (*Remaining*)
 - Time logged (*Logged*)

7.1.1.3 Estimations and the Sprint (Scrum board)

Both estimated and unestimated issues may be added into the sprint technically (Jira allows it) however, it is recommended that issues are estimated before being added into the sprint - to properly represent the value of effort being summarized. Issues in a sprint may be estimated the same way as issues that are being estimated while in the Product Backlog (via *Estimate* field). Updating the estimate on any of the items in the Sprint Backlog updates the summed Estimate at the bottom of the sprint. Also, if the total estimate differs from the time remaining (for example, because an issue was added where some work was done prior to the estimation) then the summed *Remaining* time is presented at the bottom of the sprint as well.

The screenshot shows the Jira interface. On the left, there's a sidebar with 'Sprints' and 'Epics'. The main area has a 'Backlog' section with 1 issue (TRT-6) and a 'Test sprint 1' section with 5 issues. A red box highlights the 'Estimate' column for the first three items in the sprint backlog, which are 6h, 8h, and 4h respectively. Below the sprint backlog, there are summary statistics: 5 issues, Remaining 19h, and Estimate 18h. To the right, a detailed view of 'Test story 3' is shown. This view includes fields for Status (TO DO), Priority (Medium), Component/s (None), Labels (None), Affects Version/s (None), Fix Version/s (None), Epic Link (None), Dates (Created: 56 minutes ago, Updated: 2 minutes ago), and People (Dariusz Szlek, Unassigned). A 'Description' field is also present.

The values at the bottom:

- for the **Estimate** field are the sums of all values from the *Estimate* fields of all the **user stories, tasks, and bugs** in the Sprint Backlog.
- for the **Remaining** field are the sum of all values from the *Remaining* fields of all the **user stories, tasks, bugs and their underlying sub-tasks** in the Sprint Backlog.

i Logging worktime in Jira reduces the amount of *Remaining* time which is presented for the item, where the work was logged, so: the user story, bug, task or sub-task.

! The *Estimate* time values on sub-task level does not impact the same values on the user-story (task or bug) level and vice versa. The original time estimations of sub-tasks are not reflected in their parents' *Estimate* time values, however they are reflected in their *Remaining* time values.

7.1.1.3.1 Estimating sub-tasks

Estimating sub-task issue types is possible in the same way that it is possible to estimate user stories, bugs, and tasks. However, the presentation of these estimations differs from the presentation of the estimates for the main-level issue types in a Sprint.

The estimates visible on the Backlog, concern only estimations done on the user story, bug or task level. However, the summaries below the planned sprint reflect:

- for *Estimate* - sum of all estimated time of user stories, tasks and bugs
- for *Remaining* - sum of all remaining time of user stories, tasks, bugs and their underlying sub-tasks

Information about the estimates and remaining time of sub-tasks of a user story (task or bug) are also visible in the **Backlog view** on the right-side pane view of the user story details:

The screenshot shows the Jira Backlog view. On the left, there's a sidebar labeled 'EPICS' with a dropdown arrow. Below it, under 'Test sprint 1', there are five issues: TRT-1, TRT-2, TRT-3, TRT-7, and TRT-8. A summary row indicates 'Backlog 1 issue' with 'TRT-6 Test story 4'. On the right, a detailed view of 'Test story 2' is shown. It has an estimate of 8h and remaining time of 8h, with a total remaining sum of 10.5h. This view also includes sub-tasks: ts2 subtask 1 (estimated 1.5h) and ts2 subtask 2 (estimated 1h), totaling 2.5h.

It is possible to see:

- the summarized remaining time (*Remaining sum*) for the issue and its underlying sub-tasks
- individual *Remaining* time per each sub-task along with their summarization for the current issue

Detailed information about the original and remaining time estimations (including logged work) can only be viewed in Full issue view of the sub-task (*Time Tracking* module similarly to user stories, bugs and tasks) or in the right-side pane accessed from the **Active Sprint view**:

The screenshot shows the Jira Active Sprint view for 'Test sprint 1'. The top bar indicates '9 days remaining' and 'Complete sprint'. The main area shows a board with columns: READY, IN PROGRESS, READY FOR REVIEW, and DONE. Under 'IN PROGRESS', there are two sub-tasks: 'TRT-4 ts2 subtask 1' (estimated 1.5h) and 'TRT-5 ts2 subtask 2' (estimated 1h). To the right, a detailed view of 'ts2 subtask 1' is open. It shows an estimate of 2h and remaining time of 1.5h. The right-hand sidebar contains sections for 'Log Work' and 'Attachments'.

The *Remaining* time of the individual sub-tasks can also be seen in the **Active Sprint view** on the tiles representing the sub-tasks in the swimlanes, as above.

7.1.1.3.2 Estimating and epics

It is also possible to have an overview of estimations on a per-epic basis by opening the *EPICS* tab on the left side of the Backlog view in Jira. That shows the information on:

- how much time is estimated and declared for the issues belonging to the epic
- how many issues exist in the epic and how many are estimated, unestimated and completed

The screenshot shows the Jira interface. On the left, the 'Backlog' screen displays 'EPICS' and 'All issues'. Under 'Test epic 1' (TRT-9), there are 2 issues: 'Completed' (0) and 'Unestimated' (0). The 'Estimate' field shows '14h'. Under 'Test epic 2' (TRT-10), there are 3 issues: 'Completed' (0), 'Unestimated' (2), and 'Estimate' (4h). The 'Issues without epics' section is empty. On the right, the 'Test sprint 2' screen shows 5 issues: TRT-1 (6h), TRT-2 (8h), TRT-3 (4h), TRT-7 (2h), and TRT-8 (2h). A 'Start sprint' button is visible. Below the sprint backlog, the 'Backlog' screen shows 1 issue: TRT-6 (5h). A detailed view of a story, 'Test story 3', is shown on the right, with fields for Estimate (4h) and Remaining (4h). Other sections include 'Attachments', 'Sub-Tasks' (empty), and 'Development'.

7.1.1.4 Planning and estimating a sprint in hours

In order to plan a sprint with hourly estimations in Jira, please follow the steps:

1. Take the first (most prioritized) item from the top of the backlog and enter the estimate in hours (using ways described earlier on the page)
 - a. if the item is not estimated, enter the forecasted value into the *Estimate* field
 - b. if the item is already estimated (during backlog refinement for example) but not started, then please evaluate and confirm the estimate - should the estimate change as a result, remember to update the remaining time as well to match the estimated time value
 - c. if the item is already estimated and work has been done, then please evaluate and confirm the remaining time - should the remaining time differ from the current one, update the *Remaining* time field value
2. Pull the estimated item into the created sprint
3. Repeat points 1-2 until the team's capacity has been reached, by verifying the summarization of the *Remaining* time value at the bottom of the planned sprint backlog in Jira
4. (Optional) Now, if estimates on sub-task levels are absolutely needed, then by going through every item pulled into the sprint backlog, define the estimations on sub-task during item decomposition when they are created.



Please note: for Jira reporting consistency, it is recommended to log work either on sub-task or user story level - not mixing the two approaches.

7.1.1.5 Estimating and KanBan (board)

Jira does not require nor does it present any estimation forecasts or summaries on the extended KanBan backlog and the KanBan board. KanBan is mainly considered to rely on throughput, being optimized by adapting the flow of work, hence any default metrics are not displayed and can only be configured as part of a custom view on the tiles or backlog items.

The screenshot shows the Jira configuration interface for a KanBan board. On the left, there's a sidebar with project navigation and configuration tabs. The main area displays a 'Card layout' configuration for the 'Backlog' and 'Kanban board'. Under 'Backlog', there are three cards with estimated times (8h, 2h, and None) and their respective descriptions. Under 'Kanban board', there are fields for 'Field Name' (Affects Version/s, Component/s, Time Spent) and buttons for 'Add' and 'Delete'.

Of course, **Original** and **Remaining estimates** can still be used like in the case of a regular Scrum board - the logic behind those mechanisms is the same and the fields are available during editing issues. However, without custom configuration, they will not be displayed and are not used as metrics for reporting and measurements. The estimation is also not visible on the side panel view.

The screenshot shows the Jira KanBan board view. The backlog items include EGB-2451, EGB-2448, EGB-2442, EGB-2444, EGB-2448, EGB-2395, EGB-2396, and EGB-2376. To the right, a side panel titled 'Overblik over egne sager på scanningsklienten' provides a summary of GDPR data classification (4 Personal data - RELATED, 5 Software error), threat classification (1 Low), risk analysis (None), and control status (None). It also lists people involved: Per Lerche Nielsen and Stephan Skovgård Jensen.

7.1.2 7.1.2 - Planning and starting a Sprint

The journey to starting your Sprint begins in your Jira project, on the Scrum Board representing your product or team backlog:

The screenshot shows the Jira interface for the 'Test_EG_Retail' project. The left sidebar has a 'Backlog' item selected, indicated by a blue arrow. A red box highlights the 'Backlog' section in the sidebar and the backlog list below. A green box highlights the 'Create sprint' button in the top right corner of the backlog list.

Issue Key	Description	Estimate
TRT-1	Test story 1	6h
TRT-2	Test story 2	8h
TRT-3	Test story 3	Test epic 2 (4h)
TRT-7	Test story 5	Test epic 2 (5h)
TRT-8	Test bug 1	Test epic 2 (5h)
TRT-6	Test story 4	Test epic 2 (5h)

Indicated by the red sections are your selected:

- Scrum Board (drop-down)
- Product (team) Backlog

Indicated by the green section is:

- the button which triggers the creation of a new Sprint

Indicated by the blue arrow is the selected **Backlog view** which is necessary to start a sprint (a sprint cannot be started in the **Active sprints view**).



Only an employee with a **Scrum Master** role can manage (create, start, configure and complete) a Sprint, as stated in: [7.8 - Jira & Confluence permissions schema \(see page 373\)](#)



Issues marked by a yellow color highlight and indicated with a red flag, are considered to be issues that are **Blocked** for some reason - please check the comments of the issue for more details about the impediment.

The screenshot shows the Jira interface for the 'Test_EG_Retail' project. The left sidebar has a 'Backlog' item selected. A yellow box highlights the 'Backlog' section in the sidebar and the backlog list below. A red flag icon is placed next to the 'TRT-2 Test story 2' row in the backlog list.

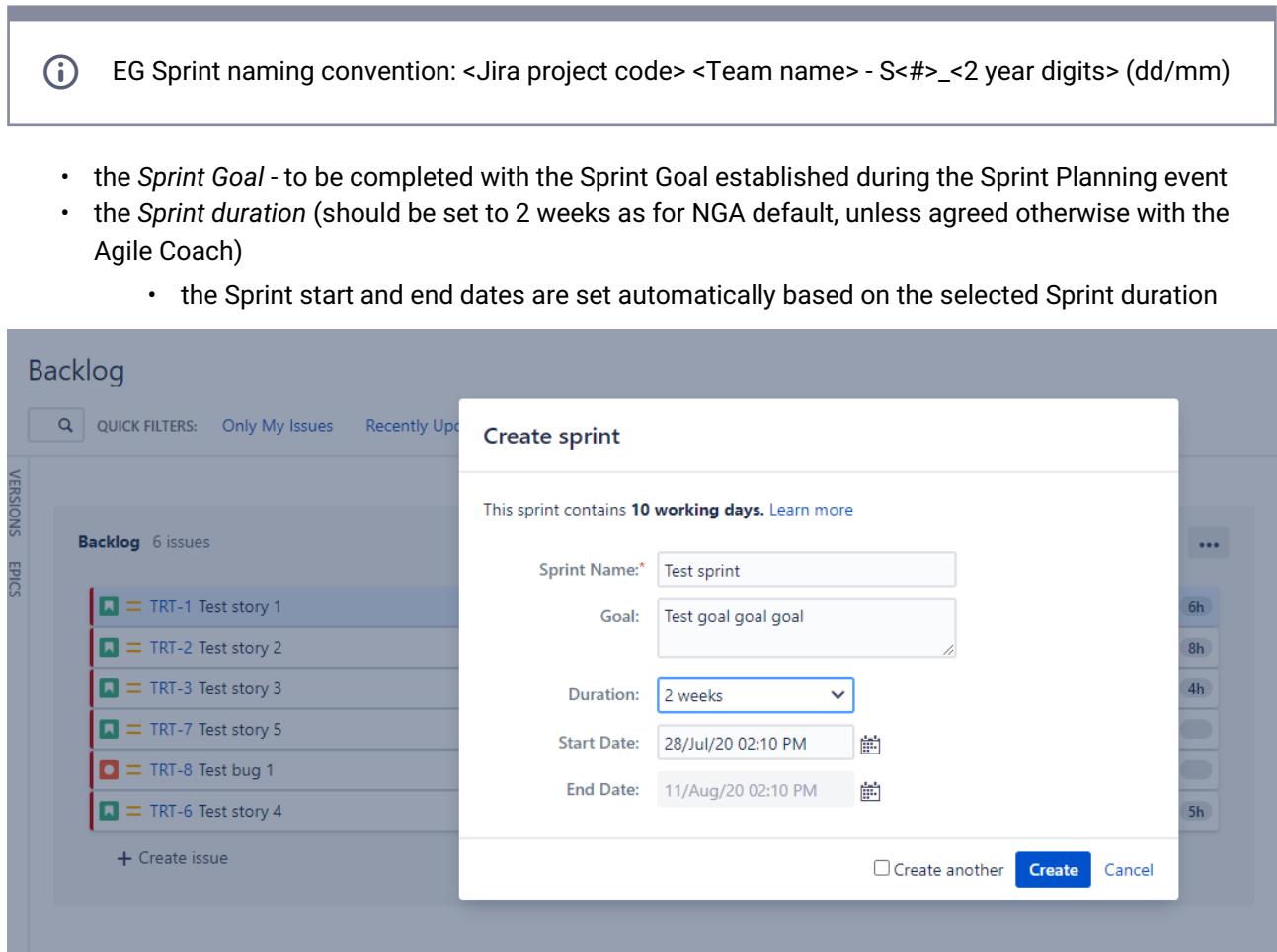
Issue Key	Description	Estimate
TRT-1	Test story 1	6h
TRT-2	Test story 2	8h
TRT-3	Test story 3	Test epic 2 (4h)
TRT-7	Test story 5	Test epic 2 (5h)
TRT-8	Test bug 1	Test epic 2 (5h)
TRT-6	Test story 4	Test epic 2 (5h)

7.1.2.1 Creating a Sprint

The first thing to do is to create an empty sprint, which can be filled with items from the Product Backlog as a part of the Sprint Planning.

In order to do that, press on the “Create Sprint” button. Once done, there is a prompt to enter the Sprint details, such as:

- the *Sprint name* - to be completed in accordance to the EG company standard as mentioned in [4.1 - What an EG cadence looks like \(see page 107\)](#)



The screenshot shows the Jira interface with the 'Backlog' tab selected. On the left, there's a sidebar with 'VERSIONS' and 'EPICS'. The main area displays a backlog of 6 issues, each with a small icon and a title: TRT-1 Test story 1, TRT-2 Test story 2, TRT-3 Test story 3, TRT-7 Test story 5, TRT-8 Test bug 1, and TRT-6 Test story 4. Below the backlog is a '+ Create issue' button. A modal window titled 'Create sprint' is open over the backlog. Inside the modal, it says 'This sprint contains 10 working days. [Learn more](#)'. The form fields are as follows:

- Sprint Name:** * Test sprint
- Goal:** Test goal goal goal
- Duration:** 2 weeks
- Start Date:** 28/Jul/20 02:10 PM
- End Date:** 11/Aug/20 02:10 PM

 There are also buttons for 'Create another', 'Create', and 'Cancel'.

Once completed, an empty sprint appears on the Jira Scrum Board:

Backlog

The screenshot shows the Jira Backlog interface. On the left, there are navigation tabs: VERSIONS, EPICS, and a selected BACKLOG tab. At the top, there's a search bar and quick filters for "Only My Issues" and "Recently Updated".

Sprint Backlog Section:

- A red box highlights a sprint named "Test sprint" containing 0 issues, created between 28/Jul/20 2:10 PM and 11/Aug/20 2:10 PM. It includes a "View linked pages" link and the text "Test goal goal goal".
- To the right of the sprint, a context menu is open with options: "Start sprint" (disabled), "Edit sprint", and "Delete sprint".
- Below the sprint, there's a "PLAN YOUR SPRINT" section with a placeholder for dragging issues and a "Create issue" button.
- At the bottom of this section, it says "0 issues Estimate 0".

Product Backlog Section:

- A red box highlights the "Backlog" section with 6 issues.
- The issues listed are:
 - TRT-1 Test story 1 (Test epic 1, 6h)
 - TRT-2 Test story 2 (Test epic 1, 8h)
 - TRT-3 Test story 3 (Test epic 2, 4h)
 - TRT-7 Test story 5 (Test epic 2, 0h)
 - TRT-8 Test bug 1 (Test epic 2, 0h)
- On the right, there are "Create sprint" and three-dot buttons.

After creating a sprint, a new section becomes visible with the Sprint name and number of contained issues. It is also possible to edit the previously entered details of the sprint, as well as delete the sprint by selecting the upper-right menu. Now it is possible to “drag & drop” items from the Product Backlog to the newly created sprint (this can also be done by right-clicking on a Product Backlog item and selecting the appropriate Sprint name in the “send to” section), as agreed during the Sprint Planning, to the amount acceptable by the team and verified against their capacity.

Backlog

QUICK FILTERS: Only My Issues Recently Updated

Test sprint 4 issues

28/Jul/20 2:10 PM • 11/Aug/20 2:10 PM View linked pages

Test goal goal goal

- TRT-1 Test story 1
- TRT-2 Test story 2
- TRT-3 Test story 3
- TRT-8 Test bug 1
- TRT-7 Test story 5

+ Create issue

Start sprint ...

4 issues Remaining 19.5h Estimate 18h

Backlog 2 issues

- TRT-6 Test story 4

+ Create issue

Backlog

QUICK FILTERS: Only My Issues Recently Updated

Test sprint 4 issues

28/Jul/20 2:10 PM • 11/Aug/20 2:10 PM View linked pages

Test goal goal goal

- TRT-1 Test story 1
- TRT-2 Test story 2
- TRT-3 Test story 3
- TRT-8 Test bug 1
- TRT-7 Test story 5

+ Create issue

Send to

- Test sprint ← (highlighted)
- Top of Backlog
- Bottom of Backlog
- Archive
- Delete
- Add flag
- Add flag and comment
- Split issue
- View in Issue navigator
- Bulk Change
- Print selected card

Start sprint ...

4 issues Remaining 19.5h Estimate 18h

Backlog 2 issues

- TRT-6 Test story 4

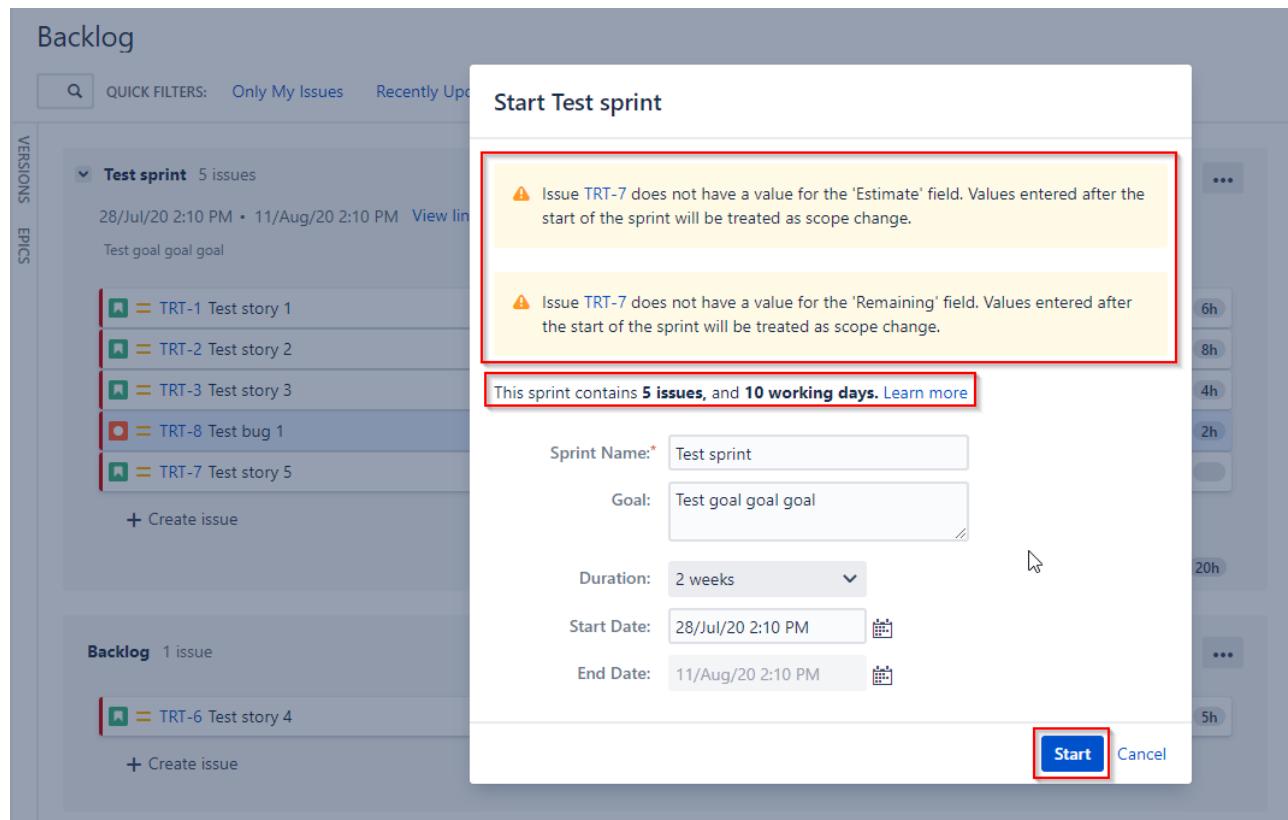
+ Create issue

7.1.2.2 Starting a Sprint

Once the Sprint Backlog is full to the understanding and acceptance of the Developers, and the following criteria are met:

- all items are estimated
- all items are decomposed
- all items meet the Definition of “Ready”

then the Sprint may be started in Jira.



By clicking the “Start Sprint” button in the upper-right corner, the presented dialog box appears with information about:

- issues in the Sprint Backlog which have not been estimated (hence the *Estimate* and *Remaining* time values are missing)
- Sprint details including the number of issues, and selected sprint duration

The sprint details may once again be verified and edited similarly to the dialog box presented upon sprint creation.

Once all parameters are properly configured, the “Start” button at the bottom of the dialog may be used to begin the Sprint in Jira.

The user is directed straight to the **Active sprints view** in scope of the same scrum board, where details regarding the ongoing Sprint can be viewed, progressed and monitored.

i Please note: a new sprint should be created but not started before the previous sprint is closed in Jira and the new sprint can only be started when the old one was closed. See: [7.1.4 - Closing a sprint \(see page 266\)](#) for information on closing a sprint.

7.1.3 Using the Active Sprint dashboard view

The **Active Sprints** view in Jira is the most commonly used dashboard in everyday work of an EG Agile Team. This dashboard provides an overview of all issues within the currently active Sprint, allowing transitioning between process states, direct access to particular Jira issues, browsing, and filtering.

7.1.3.1 Overview

In order to access the *Active Sprints* view in Jira, there is a dedicated link in the left-side pane of every project, labeled: "Active sprints".

The board consists of several minor areas of interest such as:

- Sprint name & Sprint Goal (not editable - [7.1.2 - Planning and starting a Sprint \(see page 254\)](#))
- Quick filters panel
- Remaining time till the end of the Sprint indicator

and the main part which holds all of the issues in the current sprint of the team, complying with the selected quick filters.

7.1.3.1.1 Configuring and using quick filtering

Quick Filters

Quick Filters can be used to further filter the issues in the board based on the additional JQL query.

Name	JQL	Description	
Ready issues	status = Ready	Display only issues which comply with the Definition of Ready and are ready for Sprint Planning	 
Only My Issues	assignee = currentUser()	Displays issues which are currently assigned to the current user	
Recently Updated	updatedDate >= -1d	Displays issues which have been updated in the last day	

QUICK FILTERS: Ready issues Only My Issues Recently Updated

It is possible to:

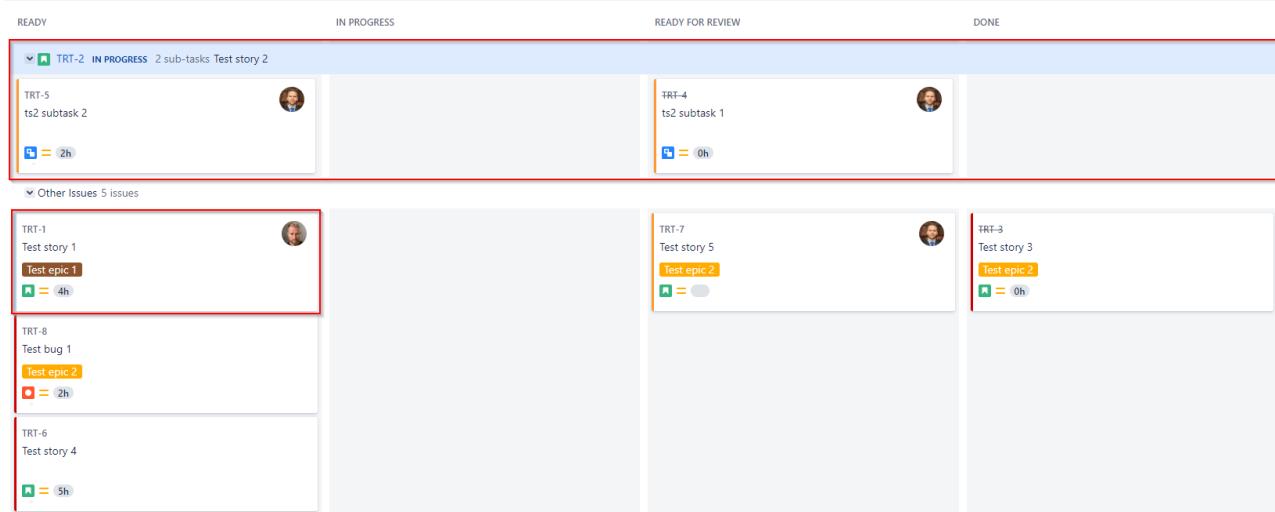
- define custom filters, which are represented by the selectable buttons on the board
- show/hide all quick filters and enable particular quick filters (quick filters are configurable)

7.1.3.2 Columns and swimlanes

The columns in the *Active Sprints* view represent statuses in the EG Jira workflow and are compliant with the workflow rules as described in: [7.1.7 - Jira issues workflow \(see page 279\)](#). Dragging an issue of type: user story, task, bug into the appropriate column changes the status of the issue to that defined in the column. Analogically dragging a sub-task into the appropriate column changes the status of that sub-task.

Changing the status of sub-tasks does not affect the status of the parent issue, therefore:

- when the first sub-task is dragged into the *In Progress* state, the parent issue should be updated with the *In Progress* status as well (this can be done by clicking on the parent issue name)
- when all sub-tasks are *Done*, the parent issue should be updated with the *Ready for Review* status, in order to indicate that the entire user story is ready to be verified by the Product Owner. A parent issue will not be able to be completed if there exist any undone sub-tasks.



There exist 2 types of swimlanes in the Jira Active sprints view:

- decomposed issue (user story, task, bug) swimlane (first selection)
- simple issues (user story, task, bug) swimlane (second selection)

The swimlanes are presented automatically depending on whether or not an issue contains sub-tasks. If an issue contains sub-tasks, then it is presented with a header of the issue and underlying sub-task tiles. Otherwise, the whole issue is presented as one tile in the “Other issues” swimlane.



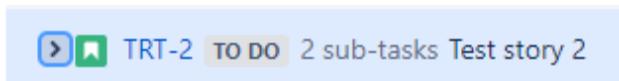
For consistency purposes and to achieve higher transparency, it is recommended for all user stories in a sprint to be decomposed with at least 1 sub-task. For issues of type bug or task, they are allowed not to be decomposed since they may represent a small piece of work on their own. All issues which are not decomposed will be visible in the “Other issues” swimlane and those of type user story should be addressed as soon as possible. The sprint backlog order is not maintained in that case between the decomposed issues and those in the “Other issues” section.

7.1.3.2.1 Assignment

Issues in progress and the next planned work items should be assigned to team members. Usually, every sub-task is assigned to one particular responsible person. In the case of user stories, which are decomposed, the assignment exists on one representative “contact” person, even if several people may be working on the work item at the time.

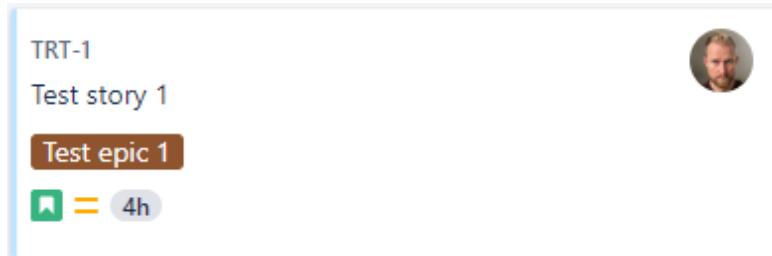
7.1.3.3 Transitioning and accessing issues

A decomposed issue header holds the following information:



- show/hide underlying sub-tasks toggle “>”
- unique issue ID
- status
- number of underlying sub-tasks
- issue summary (name)

An issue tile (independent of its type) holds the following information:



- unique issue ID
- issue summary (name)
- issue epic name
- issue type (icon)
- issue priority (icon)
- issue *Remaining* time estimate
- number of days in current status - dot indicator (when in unchanged status for several days)
- assignee avatar icon

Status change directly on the dashboard using columns can be done only on the tiles; this can be done by clicking and dragging a tile from one column to another. If a restriction exists not making it possible, the transition will not be made possible (and the target column will not light up).

For access to issues not presented as tiles on the dashboard (parent issues of sub-tasks) or for access to the details of any issue represented by a tile, it is sufficient to click it (name or id) directly to see it appear in the right-side panel to edit any necessary data.

The screenshot shows a Jira board for a 'Test sprint'. The board has four columns: READY, IN PROGRESS, READY FOR REVIEW, and DONE. In the IN PROGRESS column, there is a card for 'TRT-2' with the sub-task 'ts2 subtask 1'. To the right, a detailed view of 'Test story 2' is displayed. The story is in 'IN PROGRESS' status with an estimate of 8h and a remaining sum of 10h. It has a reporter assigned to 'Dariusz Szlek' and is unassigned. The story also has an epic link to 'Test epic 1'.

If that view is not sufficient, then the **Full page view** can be accessed either by:

- right-clicking on any *issue ID* and opening it in a new browser tab
- clicking on the *issue ID* of the issue opened in the right-side panel

After making any necessary changes and adjustments within the issue (adding comments, attachments, logging work time, etc.) it is possible to return to the full *Active Sprints* view by closing the right-side panel using the "x" in the upper right corner.

7.1.3.4 Blocking and unblocking issues

In order to mark an issue in the Sprint Backlog as **Blocked**, the user needs to:

- right-click on the issue tile, selecting “Add flag” or “Add flag and comment”
- if the option with adding a comment was selected, then insert a comment explaining the impediment and addressing responsible/accountable users

A Blocked issue is highlighted in a yellow color and marked with a red flag.



The screenshot shows a Jira board for a 'Test sprint'. The board has columns: READY, IN PROGRESS, READY FOR REVIEW, and DONE. In the READY column, there's an issue 'TRT-2' labeled 'IN PROGRESS' with 2 sub-tasks. In the IN PROGRESS column, there are several issues: 'TRT-5' (ts2 subtask 2), 'TRT-8' (Test bug 1), and 'TRT-6' (Test story 4). In the READY FOR REVIEW column, there are three issues: 'TRT-4' (ts2 subtask 1), 'TRT-7' (Test story 5), and 'TRT-3' (Test story 3). The 'TRT-7' issue is highlighted with a yellow background and a red flag icon. A context menu is open over this issue, with the 'Add flag and comment' option being selected. A modal window titled 'Add flag and comment: 1 issue' is overlaid on the board, prompting the user to add a flag to TRT-7. The modal contains a 'Comment' field with a rich text editor, a 'Visual' tab, a 'Text' tab, and a 'Viewable by All Users' checkbox. At the bottom of the modal are 'Add' and 'Cancel' buttons.

In order to resolve/remove the impediment, the user should remove the flag by:

- right-clicking on the issue tile, selecting “Remove flag” or “Remove flag and add comment”
- if the option with adding a comment was selected, then insert a comment explaining the removal of the impediment and addressing responsible/accountable users

That results in removing the yellow highlighting and red flag from the issue record but is historically recorded in the issue history.

7.1.4 7.1.4 - Closing a sprint

After a completed Sprint, the Sprint Review and Retrospective, its end should also be reflected in Jira; the task can be done only on the *Active sprints* (red) view with the properly selected *board* (blue):

The screenshot shows a Jira board titled 'Test sprint'. The sidebar on the left has a section for 'PROJECT SHORTCUTS' with links to 'Releases', 'Reports', 'Issues', and 'Components'. The main area shows a board with columns: READY, IN PROGRESS, READY FOR REVIEW, and DONE. There are several cards representing tasks. One card in the READY column is highlighted with a blue border. A red box surrounds the 'Active sprints' link in the sidebar. A blue box surrounds the 'Board' dropdown at the top right. A green box surrounds the 'Complete sprint' button at the top right.

- Indicated by the red section is the *Active sprint* view.
- Indicated by the blue section is the *board* selector, that allows you to choose the proper board (for the team or overall product) where the sprint you wish to close exists.
- Indicated by the green section is the button which triggers the closing of the existing, active sprint.



Only an employee with a *Scrum Master* role can manage (create, start, configure and complete) a Sprint, as stated in: [7.8 - Jira & Confluence permissions schema \(see page 373\)](#)

If more than 1 active Sprint exists, the user may select the one to work with:

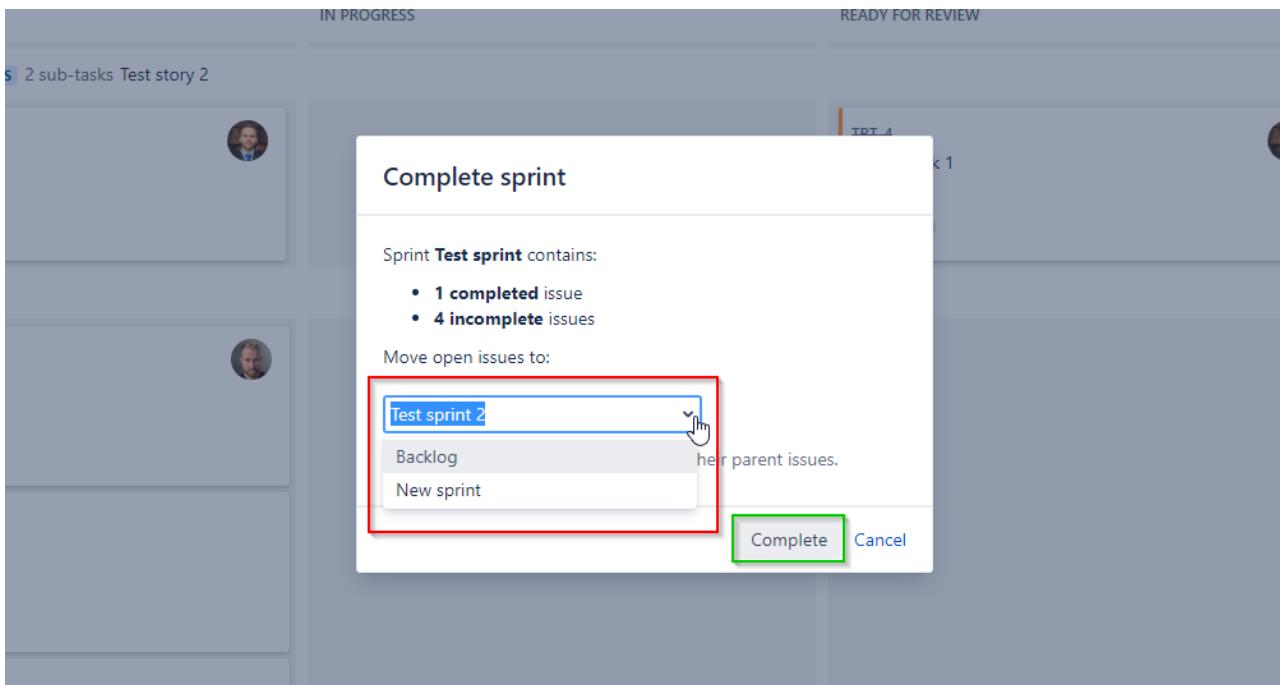
The screenshot shows a Jira board titled 'Test sprint'. The sidebar on the left has a section for 'PROJECT SHORTCUTS' with links to 'Releases', 'Reports', 'Issues', and 'Components'. The main area shows a board with columns: READY, IN PROGRESS. There are several cards representing tasks. A red box surrounds the 'Switch sprint' button in the top right corner of the board header. Below it, a dropdown menu shows 'All sprints' and 'Switch sprint'.

Otherwise, all sprints' backlog items are presented together.

7.1.4.1 Completing a sprint

Once the team has ensured that the state of the user stories, bugs and sub-tasks in the sprint is up to date the Sprint can be completed by pressing the “Complete sprint” button.

In a case, where not all Sprint Backlog items have been completed, a dialog box appears for the user to decide what to do with the remaining items:



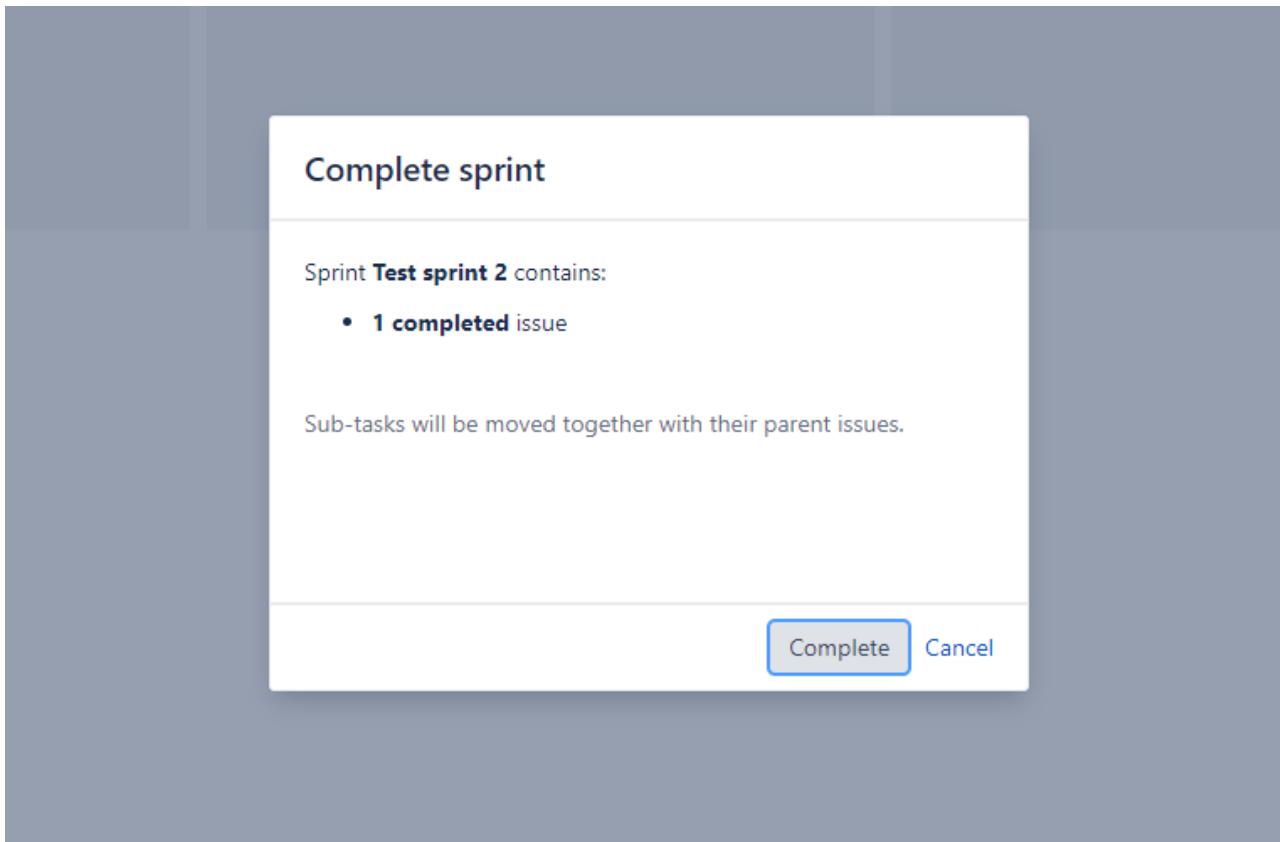
Information about the number of finished and unfinished issues is presented as a summary.

The user may choose to move the unfinished items, as indicated by the red section:

- to one of the already created, but not yet started sprints (Test sprint 2)
- to a New sprint, creating it at the same time
- to the Backlog, where they can be re-prioritized and re-planned in the future

The user confirms the Sprint completion by pressing the “Complete” button or cancels the action by pressing “Cancel”, as indicated by the green section.

If all items in the Sprint Backlog have been completed, the user does not get a choice, but only a confirmation of the action:



(i) Please note: a new sprint should be created but not started before the previous sprint is closed in Jira and the new sprint can only be started once the old one was closed. See: [7.1.2 - Planning and starting a Sprint \(see page 254\)](#) for information on creating and starting a sprint.

7.1.5 Using the KanBan board and extended backlog

In the case of Agile teams with a more maintenance approach or whenever deemed more valuable for the team to use the KanBan method, it is possible to configure a KanBan board in the Jira project. The core elements of the boards are very similar ([N2 - 7.1.3 - Using the Active Sprint dashboard view - Next Generation Agile - Confluence EG A/S⁴⁶](#)) between KanBan and Scrum however, there are some differences and specifics to such a board type, that will be described in this chapter.

⁴⁶ <https://confluence.eg.dk/display/NG/N2++7.1.3++Using+the+Active+Sprint+dashboard+view>

7.1.5.1 The KanBan extended backlog

The basic KanBan backlog is a single list of all items that are created in the project. All issue types **including** epics are listed in the backlog - as compared to the Scrum backlog where the epics are treated more like a filter and are available in the left slideout panel. The KanBan backlog in the basic form is just the KanBan board where all queued items and all active items are visible on one page. This is a very simple configuration but can be problematic with control over a large backlog that is full of work forecasted for the not so near future.

The screenshot shows a Jira KanBan board titled "Kanban board". The board has four columns: "READY 1125", "IN PROGRESS 121", "NEW COLUMN 121", and "DONE 0 of 4635". The "READY" column contains several issues, including A4-193, A4-355, A4-6034, A4-3057, A4-2559, and A4-8225. The "IN PROGRESS" column contains A4-155 and A4-7109. The "NEW COLUMN" and "DONE" columns are currently empty. On the left side, there is a sidebar with navigation links for "Kanban board", "Releases", "Reports", "Issues", "Components", and "Capacity Tracker". A red box highlights the "Kanban board" link in the sidebar.

Issues with sub-tasks can be listed underneath their parent story, task or bug (depending on board configuration for the swimlanes), and also epics are visible in the backlog/board.

It is possible to split this view into 2 - similarly to the Scrum board, where a separate backlog and separate KanBan board are visible. This allows defining a single initial status where all items, which are not ready for development are stored in a separate compartment, while all items with the next and following statuses are visible in the KanBan board. That way the KanBan board is more readable and not cluttered with future tasks.

The screenshot shows the "Configure A4 KanBan" screen. Under the "CONFIGURATION" tab, the "Columns" section is selected. It shows a "Kanban backlog" card with a red border. Below it, there are four status boxes: "Ready" (162 issues), "In Progress" (136 issues), "New Column" (146 issues), and "Done" (6531 issues). Each status box has a "No Min" and "No Max" constraint. To the right, there is a "Unmapped Statuses" section. A red box highlights the "Kanban backlog" card and the status boxes.

It is possible to define which status is to be part of the KanBan backlog (it is usually the initial one "To Do" and the following status which will be the entry point for issues deemed as ready for development).

Also, it is possible to:

- disable/enable epics as a panel - they can be issues in the backlog instead if preferred, like in the simple KanBan board
- define minimum and maximum W(ork) I(n) P(rogress) limits for optimizing the KanBan flow ([N2 - 5.7](#))
 - [Using team metrics - Next Generation Agile - Confluence EG A/S⁴⁷](#)

The result is as follows:

Ready 151 issues

- A4-7911 401 - RDV/GIT/Netkons uddannelse og implementering i Finance
- A4-7911 405 - Password til Nets som password til Skat (Masterkey)
- A4-8383 EG - FR: Mock-up
- A4-8384 EG - FR: Transparens Beskrivelse og design
- A4-8385 EG - FR: Automatisk - Beskrivelse og Design
- A4-8381 EG - FR: Transparens, Modul til sammenligning af bookninger indenfor interval
- A4-8382 EG - FR: Transparens, Robot modul til at fjerne markering der er blevet for gamle
- A4-8386 EG - FR: Transparens, Markering på skærmblade
- A4-8392 EG - FR: Transparens, Modul der finder dem der ligner mig ved visning
- A4-8387 EG - FR: Transparens, Vis andre bookinger der ligner
- A4-8371 EG - FR: Konsolidering 7535 samletstale/opdateringer
- A4-8369 EG - FR: Konsolidering 7535 Åben/Luk/options
- A4-7503 EG - indscanning fragtbevægelse via ABC
- A4-7504 EG - Sænring - VR*
- A4-8011 SFI ETA brug af addeEta api i egen kode
- A4-8010 SFI ETA autocreate logic - æverdig GO - FASE II
- A4-7184 Future Planning - Overordnet fase 1-3 - bruges ikke aktuelt

The KanBan backlog is very similar to the Scrum backlog. The main differences include:

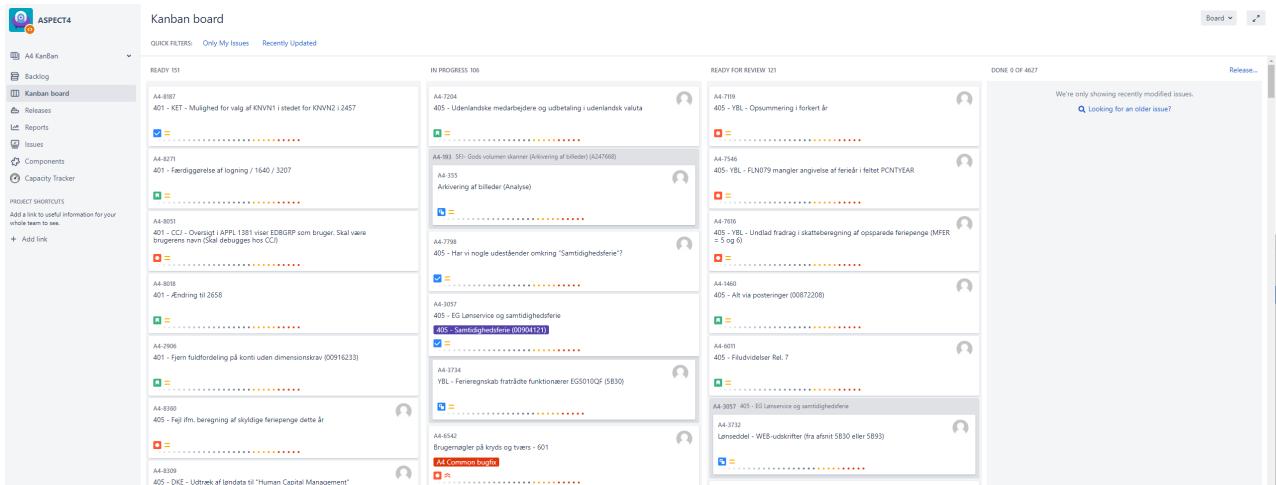
- lack of sprints
- a dedicated section for the "next status - Ready (for development)" - items which are already visible on the KanBan board but also on the KanBan backlog, as those issues which are ready to be worked on but the work has not yet started (in progress)
- lack of estimations (hourly or story points)

7.1.5.2 The KanBan board

When working with the KanBan method, the KanBan board is the most commonly used dashboard of an EG Agile Team. This dashboard provides an overview of all issues which are known and which have met the Definition of Ready, as well as their progress and assignment. If the KanBan backlog is disabled, then all issues including the ones which are not yet ready for development appear on the board; it is recommended however to use the KanBan backlog to avoid congestion and an overwhelming number of issues appearing on the board. All of the items on the KanBan board can be prioritized to align the sequence of development according to the expectations.

The KanBan board also utilizes the defined WIP limits to illustrate the bottlenecks in the process in order to optimize the flow of the work by the Agile team.

⁴⁷ <https://confluence.eg.dk/display/NG/N2++5.7++Using+team+metrics>



7.1.5.2.1 Overview

In order to access the KanBan board, there is a link on the left-side pane of every project, labeled "Kanban board". There is a configurable quick filter panel available, similarly to the Scrum board, however the Sprint data and estimations on the cards do not show.

The columns and swimlanes follow the same logic to that of the Scrum board: [N2 - 7.1.3 - Using the Active Sprint dashboard view - Next Generation Agile - Confluence EG A/S⁴⁸](#) and can differ according to the configuration of the swimlanes:

- Stories - grouping of sub-tasks under a user story
- No swimlanes - no grouping (selected above)
- and other less frequently used

7.1.5.2.2 Operating on issues on the KanBan board

Similarly to the Scrum boards, it is possible to:

- transition issues by dragging them into a particular status
- blocking / unblocking issues (flagging)
- prioritizing issues

The following options are possible by context-clicking on the selected issue or by performing a drag operation; details described in scope of: [N2 - 7.1.3 - Using the Active Sprint dashboard view - Next Generation Agile - Confluence EG A/S⁴⁹](#)

7.1.5.2.3 Completed issue display time

Since the KanBan board can become overloaded with old issues, there is a cleanup mechanism in place to hide those **completed** items which have been finished longer than a specified time ago:

⁴⁸ <https://confluence.eg.dk/display/NG/N2++7.1.3++Using+the+Active+Sprint+dashboard+view>

⁴⁹ <https://confluence.eg.dk/display/NG/N2++7.1.3++Using+the+Active+Sprint+dashboard+view>

Configure A4 KanBan

The screenshot shows the 'General and filter' configuration for the 'A4 KanBan' board. It includes fields for Board name (A4 KanBan), Administrators (Dariusz Szlek), Saved Filter (Filter for A4 KanBan), Ranking (Using Rank), Projects in board (ASPECT4), and a Kanban board sub-filter for unreleased work. A dropdown menu for 'Hide completed issues older than' is open, showing options: 2 weeks (selected), 1 week, 2 weeks, 4 weeks, and Show all.

General and filter

The Board filter determines which issues appear on the board. It can be based on one or more projects, or custom JQL defined here.

General

Board name **A4 KanBan**

Administrators **Dariusz Szlek (xxagz@eg.dk)**

Filter

Saved Filter **Filter for A4 KanBan**
[Edit Filter Query](#)

Shares **Project: ASPECT4 (VIEW)**
[Edit Filter Shares](#)

Filter Query **project = A4 ORDER BY Rank ASC**

Ranking **Using Rank**

Projects in board **ASPECT4**
[View permission](#)

Kanban board sub-filter
fixVersion in unreleasedVersions() OR fixVersion is EMPTY
Further filtering of issues for unreleased work.

Hide completed issues older than

- (Selected)
-
-
-
-

Atlassian Jira | sha1:36e93c7 | [About Jira](#) · [Report a problem](#)

This Jira site is for non-production use only.

ATLASSIAN

This will make the board more readable and transparent with time, as the team progresses and more consecutive items are completed that do not disappear with the completed Sprint as it is in case of the Scrum board.

7.1.6 - Jira issue types and hierarchy

7.1.6.1 Jira issue types

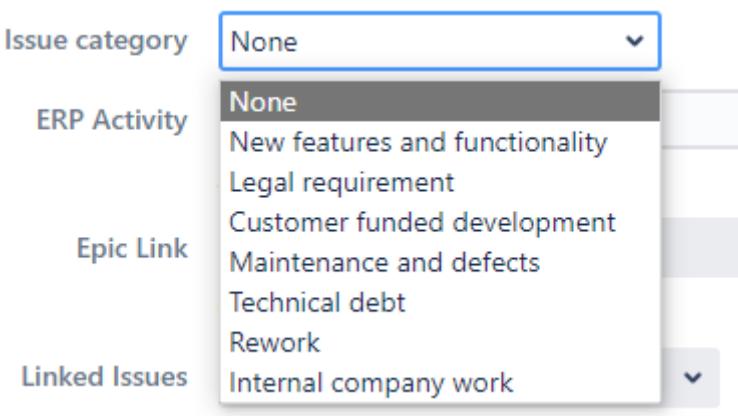
- **Epic** - a set of requirements, a big user story that needs to be broken down.
- **Bug** - a bug is a problem that impairs or prevents the functions of a product.
- **User story** - a user story is the unit of work that needs to be done. Represents the business requirement.
- **Task** - a task represents work that needs to be done and it's not related to any user story. Like participation in training or actions from a retrospective.

- **Subtask** - a subtask is the smallest piece of work that is required to complete a story or task.
- **Business project** - a container for a selected subset of issues (epics, user stories) that defines a specific goal, project and/or a distinguished budget. Can be spanned across multiple projects in Jira.

i Every **user story**, **task** and **bug** will have a mandatory custom drop-down field „Issue category“ with the necessity to select one of the following values:

- New features and functionality
- Legal requirement
- Maintenance and defects
- Technical debt
- Customer funded development
- Rework
- Internal company work (should only be used for *Task* issue types)

Sub-tasks in the abovementioned issue types will inherit their parents' issue category by default but will be editable if needed, later on.



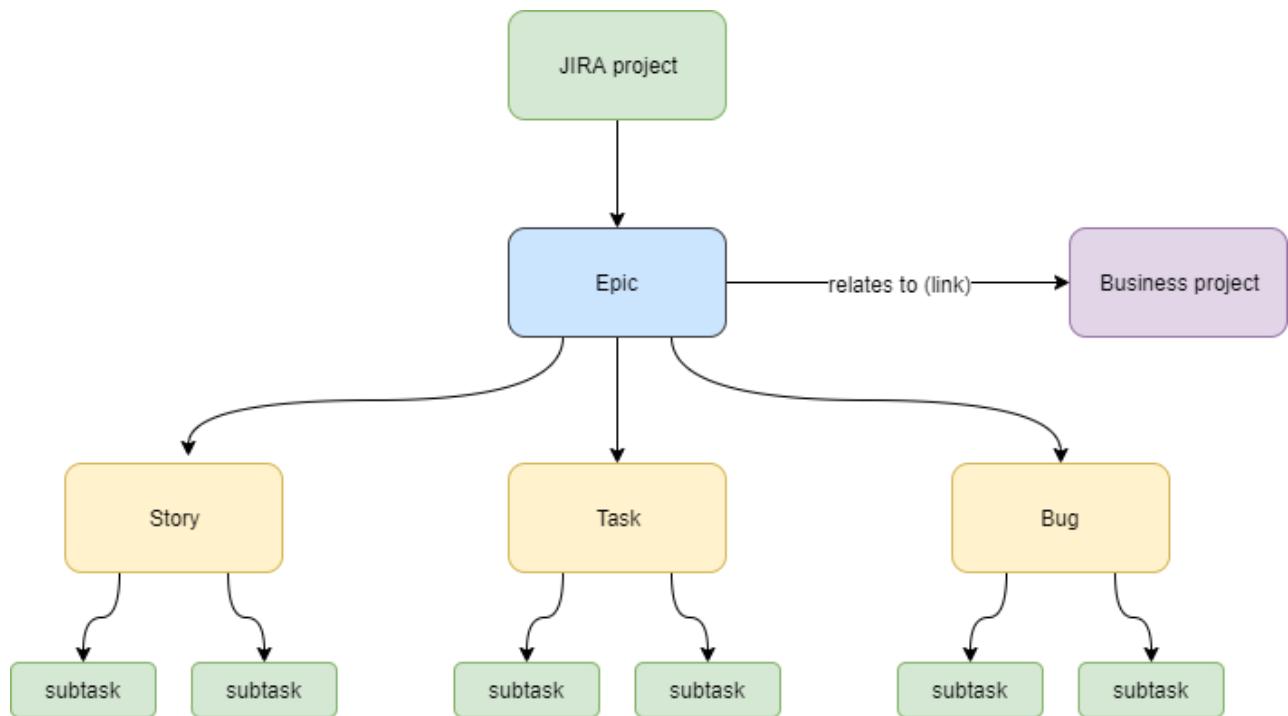
7.1.6.1.1 Issue category descriptions

Category	Description	Recommended % distribution
New features and functionality	Strategic development of functions, functionality and tech initiatives that are justified by the market	60
Legal requirement	Features or functionalities required as a result of legal adjustments or other governmental originated requirements	Low

Category	Description	Recommended % distribution
Customer funded development	Customer-specific development of features and functionalities that are funded by a customer	Low
Maintenance and defects	<p>The main purpose of software maintenance is to modify and update software products after delivery.</p> <p>Corrective maintenance:</p> <p>Reactive modification of a software product performed after delivery to correct discovered problems. Corrective changes in software maintenance are those that fix bugs, flaws and defects in the software.</p> <p>Adaptive maintenance:</p> <p>Modification of a software product performed after delivery to keep a software product usable in a changed or changing business and technical environment.</p> <p>Environment refers to the conditions that influence the software from the outside. E.g. Change in business rules, policies and laws and response to new operating systems, new hardware, and new tech platforms, to keep the program compatible. Adapting to GDPR, Medcom standards, legislation, etc. are examples of this.</p> <p>Perfective maintenance:</p> <p>Modification of a software product after delivery to improve performance or maintainability. It involves making functional enhancements to the system in addition to the activities to increase the system's performance even when the changes have not been suggested by faults.</p>	20

Category	Description	Recommended % distribution
Tech debt	<p>Deliver now and fix it later is a popular approach because teams can reduce the time it takes to get a product to market. It's also great because software engineers don't have to spend too much time developing something that might be used. In order to make this approach work, continuous refactoring must be done, else technical debt will accumulate, and it will be hard to add new features in the future.</p> <p>Primary activities are cleaning up and simplifying code, make it understandable to others, without changing its behavior. Also known as refactoring of code that is duplicated, using ambiguous variable names, unused variables – methods or classes.</p> <p>The Tech debt category includes Preventive maintenance activities to detect and correct latent faults in the software product before they become effective faults. It comprises documentation updating, code optimization, and code restructuring, reengineering of the legacy codebase and converting its structure or converting to a new language.</p>	20
Rework	Rework due to bad design or quality in functional and non-functional requirements specifications.	Low
Internal company work	<p>This Category is used for work that cannot be related to a specific product or customer.</p> <p>The category covers :</p> <ul style="list-style-type: none"> • Sprint Retrospectives • Internal projects (That are EG oriented and not related to our product and services that we develop and deliver to our customers. Eg process improvement projects, implementation, and adoption of new EG systems, data migration to new Jira platform, etc.) • Education <p>The following is NOT covered by this category:</p> <ul style="list-style-type: none"> • Sickness • Leave • Absence • Holiday • Others and administrative hours 	Low

7.1.6.2 Jira issues hierarchy



7.1.6.3 Jira bug issue resolution types

Working with issues of type *Bug* introduces an additional step to identify the resolution type of the completed defect. When finalizing a bug into status *Done*, an additional window appears in Jira in order to specify in more detail how the defect was resolved.

Test_EG_Retail / TRT-8

Test bug 1

[Edit](#) [Comment](#) [Assign](#) [More](#) [Needs Work](#) [Done](#)

[Details](#)

Type:	<input checked="" type="radio"/> Bug	Status:	READY FOR REVIEW (View Workflow)
Priority:	<input type="radio"/> Medium	Resolution:	Unresolved
Labels:	None		

[Field Tab](#) [GDPR](#)

Epic Link: [Test epic 2](#)

Sprint: Test sprint

Issue category: Maintenance and defects

Include in client release notes: N/A

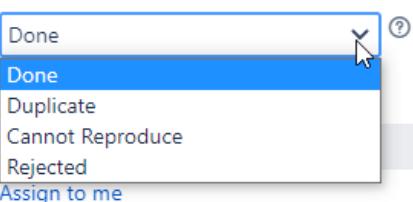
Include in technical release notes: N/A

[Description](#)

Click to add description

Among other things like adding a comment, assignee or logging work time, there is a mandatory step of selecting the *Resolution* from the existing drop-down input.

Done

Resolution* Done 

Fix Version/s

Assignee

Time Spent (eg. 3w 4d 12h) 

Date Started 29/Jul/20 4:09 PM 

Remaining Estimate Adjust automatically
 Use existing estimate of 2h
 Set to (eg. 3w 4d 12h)
 Reduce by (eg. 3w 4d 12h)

ERP Activity
AX Activity ID

Comment 

 Visual  Text    Viewable by All Users

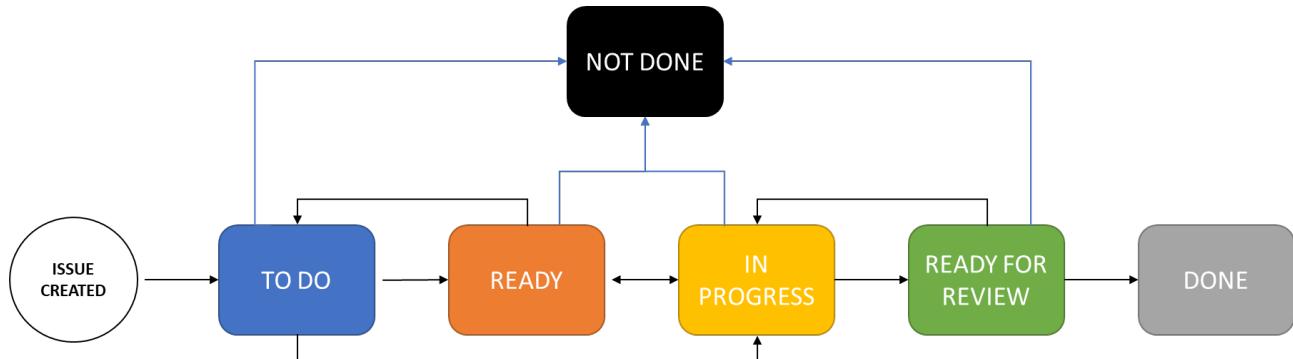
Done **Cancel**

The possible options for bug resolution are:

- **Done** - for defects, which have been fixed or otherwise resolved
- **Duplicate** - for defects, which have been identified as duplicates of an already existing issue
- **Cannot Reproduce** - for defects, which were not possible to be reproduced on the reported environment, given the provided data and/or description
- **Rejected** - for defects, which have been deemed as unjustified or otherwise not considered a bug

7.1.7 7.1.7 - Jira issues workflow

7.1.7.1 Default Jira issues workflow (All issue types)



7.1.7.1.1 Status descriptions and possible transitions

STATUS	Transition possible	Transition NOT possible	Description
TO DO	READY, IN PROGRES S, NOT DONE	READY FOR REVIEW, DONE	This is the initial status for Jira issues - upon creation this status is set by default. This is the status for all items, which are not prepared well enough to be worked on or no work has been done so far to make them ready.
READY	TO DO, IN PROGRES S, NOT DONE	READY FOR REVIEW, DONE	This status indicates that a Jira issue is ready for work within a sprint. For user stories it usually signifies that the Definition of Ready has been met, it is initially estimated and the Scrum Team sees no further topics which may hold back the work. For bugs it signifies that the defect has been properly described and analysis, as well as fixing if necessary can begin.
IN PROGRES S	READY, READY FOR REVIEW, NOT DONE	TO DO, DONE	This status indicates that work on a Jira issue is currently being done. The scope of work defines all needed tasks including design, development and testing to be done by the development team and should cover the progress up until delivery of a functional demo. The development team needs to define all of the work needed to be done by them to complete the Jira issue.

STATUS	Transitio n possible	Transition NOT possible	Description
READY FOR REVIEW	DONE, IN PROGRES S, NOT DONE	TO DO, READY	This status serves as a checkpoint for the Jira issue to be verified and confirmed by the Product Owner against the Definition of Done, the Acceptance Criteria and his/her expectations. It is a chance to confirm the declaration of the team that the Jira issue is finished as expected. Issues in this status require the PO to approve and accept the user story or reject it with appropriate comments to be improved.
DONE	-	ANY	This is a terminal status that signifies a user story has been completed and accepted by the Product Owner (or his/her delegate). Once this status has been set, it may not be reverted - code has been merged to the appropriate branch. Should any additional changes be needed at this point, a new Jira issue should be created.
NOT DONE	-	ANY (EXCEPT DONE)	This is a terminal status that signifies a user story has not been entirely completed or will not be completed, has been rejected or otherwise marked as deprecated or irrelevant by the Product Owner. Should this become desired in the future a new Jira issue should be created.

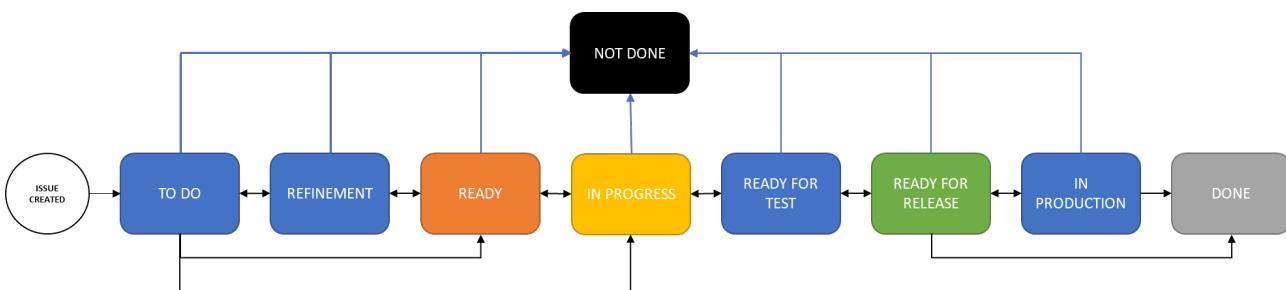
7.1.7.1.2 Transitions

From status	To Status	Remarks
TO DO	READY	The transition takes place, when all necessary conditions have been met for a Jira issue, confirming that work on a Jira issue can be started (acceptance criteria, DoR, other pre-requisites)
READY	TO DO	If the conditions previously met have become obsolete or if it has been identified that these conditions were falsely assumed to have been met, the reverse transition takes place to indicate the lack of readiness for work within a Sprint.

From status	To Status	Remarks
TO DO	IN PROGRESS	It is possible in some specific cases that the Jira issue READY quality gate is skipped and work can be started on an issue directly. Such cases include: simple bugs, where a straightforward description is sufficient and provided on bug creation or sub-tasks, which created by the development team represent small pieces of work to be done on a daily basis and do not require complex descriptions.
READY	IN PROGRESS	The transition takes place for those Jira issues, already in READY state, where the work has been started or is starting briefly. This is an indication that an issue is no longer planned or waiting for its turn to be processed, but actual design/implementation/testing tasks are being conducted by the development team.
IN PROGRESS	READY	In case the sprint work has been descoped from a sprint, postponed or simply stopped / discontinued then the status can be reversed back to the READY status. Special caution needs to be maintained to when replanning the issue to ensure that all of the already completed work has been identified and no waste is introduced into the code.
IN PROGRESS	READY FOR REVIEW	If a Jira issue is deemed completed by the development team, this transition takes place. It indicates that by the team's understanding of the expectations, acceptance criteria and definition of done, the item is finalized and releasable. All code reviews, documentation, testing, etc. have been completed and delivered; the Jira issue is ready to be confirmed as is, by the Product Owner or his/her delegate.
READY FOR REVIEW	IN PROGRESS	Upon verification of a Jira issue, if any errors, problems or deviations from the Product Owner's expectations have been identified, then this transition takes place, returning the Jira issue to the IN PROGRESS state for further work / changes / improvements to be done by the development team. The Product Owner should indicate along with this transition in comment or direct communication, his/her remarks.

From status	To Status	Remarks
READY FOR REVIEW	DONE	<p>Upon verification of a Jira issue, if all conditions of DoD, as well as the acceptance criteria and Product Owner's expectations in general have been met, then this transition takes place. It indicates that the Jira issue has been successfully completed and can be delivered in the existing form to the target environment or to the customer. No major remarks in scope exist and if any additional changes out of scope exist, they are reflected in the Product Backlog with a separate Jira issue.</p> <div style="border: 2px solid #f0adbd; padding: 10px; margin-top: 10px;"><p> Note:</p><ul style="list-style-type: none">Only Product Owner can make this transition for stories and bugs (on tasks and on sub-task level PO role is not needed).All sub-tasks of parent issue must be closed before transitioning from Ready for Review to Done status</div>
*ALL	NOT DONE	The transition takes place from all statuses (except DONE) whenever a Jira issue is considered to be deprecated, rejected, dismissed or otherwise considered irrelevant, and should be closed but not deleted. The transition is terminal, meaning the status cannot be reversed upon submission.

7.1.7.2 Alternative Jira issues workflow (Story, Task, Bug, Sub-Task issue types)



7.1.7.2.1 Status descriptions and possible transitions (alternative workflow only)

Status	Transition possible	Description
REFINEMENT	TO DO, READY, NOT DONE	This status indicates that an issue is ready for or undergoing backlog refinement activities in preparation for future planning. This status differentiates the item in the backlog from other items in backlog, which are only maintained there for work being done in the future.
READY FOR TEST	IN PROGRESS, READY FOR RELEASE, NOT DONE	This status indicates that an issue is ready to be processed through various levels of quality assurance tools and processes as a final step in ensuring the completion of a backlog item in development.
READY FOR RELEASE	READY FOR TEST, IN PRODUCTI ON, NOT DONE	see READY FOR REVIEW for Jira default workflow
IN PRODUCTION	READY FOR RELEASE, DONE, NOT DONE	This status indicates that an issue has been released into production, but may still require some work before it can be considered as done. It may also represent the need for deployment to multiple customer instances as part of completing the whole item.

7.1.7.2.2 Transitions

From status	To Status	Remarks
TO DO	REFINEMENT	The transition takes place, when backlog refinement activities are ready to be started or have been started on a Jira issue; work is ready to be executed to prepare the item for future planning.
TO DO	READY	The transition takes place, when all necessary conditions have been met for a Jira issue, confirming that work on a Jira issue can be started (acceptance criteria, DoR, other pre-requisites)

From status	To Status	Remarks
TO DO	IN PROGRESS	It is possible in some specific cases that the Jira issue READY quality gate is skipped and work can be started on an issue directly. Such cases include: simple bugs, where a straightforward description is sufficient and provided on bug creation or sub-tasks, which created by the development team represent small pieces of work to be done on a daily basis and do not require complex descriptions.
REFINEMENT	TO DO	This is a rollback transition stopping or postponing backlog refinement activities on a Jira issue until further agreement.
REFINEMENT	READY	The transition takes place when all necessary conditions have been met for a Jira issue, confirming that work on a Jira issue can be started (acceptance criteria, DoR, other pre-requisites)
READY	REFINEMENT	This is a rollback transition reverting an item to backlog refinement once an initially READY item has been deemed not ready and requires additional refinement work to be conducted in order to meet the Definition of Ready
READY	IN PROGRESS	The transition takes place for those Jira issues, already in READY state, where the work has been started or is starting briefly. This is an indication that an issue is no longer planned or waiting for its turn to be processed, but actual design/implementation/testing tasks are being conducted by the development team.
IN PROGRESS	READY	In case the sprint work has been descoped from a sprint, postponed, or simply stopped / discontinued then the status can be reversed back to the READY status. Special caution needs to be maintained when replanning the issue to ensure that all of the already completed work has been identified and no waste is introduced into the code.
IN PROGRESS	READY FOR TEST	This transition takes place for those Jira issues, which are ready to undergo any defined quality assurance activities that fall out of scope of those activities conducted during the development of the item.
READY FOR TEST	IN PROGRESS	This is a rollback transition for those Jira issues which ceased quality assurance activities to reassess or revert to development activities before further conduction of any required QA activities on the Jira issue.

From status	To Status	Remarks
READY FOR TEST	READY FOR RELEASE	If a Jira issue is deemed completed by the development team, this transition takes place. It indicates that by the team's understanding of the expectations, acceptance criteria and definition of done, the item is finalized and releasable. All code reviews, documentation, testing, etc. have been completed and delivered; the Jira issue is ready to be confirmed as is, by the Product Owner or his/her delegate.
READY FOR RELEASE	READY FOR TEST	Upon verification of a Jira issue, if any errors, problems or deviations from the Product Owner's expectations have been identified, then this transition takes place, returning the Jira issue to the READY FOR TEST state for further quality assurance activities to be done by the team. The Product Owner should indicate along with this transition in a comment or direct communication, his/her remarks.
READY FOR RELEASE	IN PRODUCTION	This transition takes place upon deployment of the issue source code to a production environment, but still requires further work (such as configuration) or other deployments to take place in scope of the same feature but multiple customer instances.
READY FOR RELEASE	DONE	Upon verification of a Jira issue, if all conditions of DoD, as well as the acceptance criteria and Product Owner's expectations in general, have been met, then this transition takes place. It indicates that the Jira issue has been successfully completed. No major remarks in scope exist and if any additional changes out of scope exist, they are reflected in the Product Backlog with a separate Jira issue.
IN PRODUCTION	READY FOR RELEASE	This is a rollback transition indicating that an issue needs to be reverted or withdrawn from a production deployment. The solution may be as well neutralized to introduce all necessary improvements as stated by the Product Owner.
IN PRODUCTION	DONE	Upon verification of a Jira issue, if all conditions of DoD, as well as the acceptance criteria and Product Owner's expectations in general, have been met, then this transition takes place. It indicates that the Jira issue has been successfully completed. No major remarks in scope exist and if any additional changes out of scope exist, they are reflected in the Product Backlog with a separate Jira issue.
*ALL	NOT DONE	The transition takes place from all statuses (except DONE) whenever a Jira issue is considered to be deprecated, rejected, dismissed or otherwise considered irrelevant, and should be closed but not deleted. The transition is terminal, meaning the status cannot be reversed upon submission.

7.1.7.2.3 Blocking and unblocking issues in the Product Backlog

This allows users to indicate that there is a problem with a Jira issue that cannot be handled by the Development or Scrum Team itself. It raises attention to the item for a fast resolution of the impediment, in order to allow further progress of work.

Issues that are blocked are labeled by a yellow highlight and a red flag on the issue record.

The screenshot shows a Jira backlog with the following items:

- TRT-1 Test story 1
- TRT-2 Test story 2** (highlighted with yellow background and red flag)
- TRT-3 Test story 3
- TRT-8 Test bug 1
- TRT-6 Test story 4

To the right of the backlog, there are sections for "Test epic 1" (6h), "Test epic 1" (8h), "Test epic 2" (4h), "Test epic 2" (2h), and "5h".

In order to mark an issue in the Product Backlog as **Blocked**, the user needs to:

- right-click on the issue, selecting (1) “Add flag” or (2) “Add flag and comment”
- if (2) was selected, then also insert a comment explaining the impediment and addressing responsible/accountable users

The screenshot shows the context menu for issue TRT-2 in the backlog. The "Add flag" option is highlighted with a yellow background and a red flag icon. Other options in the menu include "Send to", "Top of Backlog", "Bottom of Backlog", "Archive", "Delete", "Add flag and comment", "Split issue", "View in Issue navigator", "Bulk Change", and "Print selected card".

The screenshot shows the "Add flag and comment" dialog box. The "Add flag to TRT-2" option is selected. The dialog includes a "Comment" text area with rich text editing tools, a "Visual" tab, a "Text" tab, and a "Viewable by All Users" checkbox. At the bottom are "Add" and "Cancel" buttons.

In order to resolve / remove the impediment, the user should remove the flag by:

- right-clicking on the issue, selecting (1) “Remove flag” or (2) “Remove flag and add comment”
- if (2) was selected, then also insert a comment explaining the removal of the impediment and addressing responsible/accountable users

7.1.8 7.1.8 - Personalized MS Outlook rule for Jira notifications

Jira On-Premise can generate quite a load of notifications associated with user actions, defined by the fact that a user has become a watcher of an issue or the user is the creator of an issue. If the user is the creator of an issue, he/she is not able to remove notifications for special events like someone has logged work to AX for the specific issue. To filter these specific type of events, see sections for Creating a filter rule in MS Outlook.

7.1.8.1 Jira Notification Scheme (Default setting)

This is the events that triggers a notifications in Jira, combined with who gets the notification.

Events	Description	All Watchers	Current Assignee	Reporter
Issue Created	An issue has been entered into the system.	X	X	X
Issue Updated	An issue has had its details changed. This includes the deletion of an issue comment.	X	X	X
Issue Assigned	An issue has been assigned to a new user.	X	X	X
Issue Resolved	An issue has been resolved (usually after being worked on and fixed).	X	X	X
Issue Closed	An issue has been closed. (Note that an issue may be closed without being resolved).	X	X	X
Issue Commented	An issue has had a comment added to it.	X	X	X
Issue Comment Edited	An issue's comment has been modified.	X	X	X
Issue Comment Deleted				
Issue Reopened	An issue has been re-opened.	X	X	X

Issue Deleted	An issue has been deleted.	X	X	X
Issue Moved	An issue has been moved into or out of this project.	X	X	X
Work Logged On Issue	An issue has had hours logged against it (i.e. a worklog has been added).	X	X	X
Work Started On Issue	The Assignee has started working on an issue.	X	X	X
Work Stopped On Issue	The Assignee has stopped working on an issue.	X	X	X
Issue Worklog Updated	An entry in an issue's worklog has been modified.	X	X	X
Issue Worklog Deleted	An entry in an issue's worklog has been deleted.	X	X	X
Generic Event	The exact nature of this event depends on the workflow transition(s) ⁵⁰ from it was fired.	X	X	X
Issue Archived		X	X	X
Issue Restored		X	X	X

7.1.8.2 Stop watching an issue

Stop watching a specific issue. Issues that you are watching, can be unwatched, by selecting "Stop watching this issue" in the issue. See screenshot below

⁵⁰ <https://confluence.atlassian.com/adminjiraserver0810/working-with-workflows-1014674081.html>

The screenshot shows a Jira issue page for 'Test story 1'. At the top, there's a navigation bar with 'Edit', 'Comment', 'Assign', 'More', 'Not Ready For Development', 'Start Progress', and 'Admin' buttons. Below the navigation is a 'Details' section with fields like 'Type: Story', 'Priority: Medium', 'Status: READY (View Workflow)', and 'Resolution: Unresolved'. Under the 'Field Tab' tab, there are fields for 'Epic Link: Test epic 1', 'Sprint: Test sprint', 'Issue category: New features and functionality', and 'Include in client release: N/A'. On the right side, there's a 'People' section showing 'Assignee: Jaroslaw Krochmalski', 'Reporter: Dariusz Szlek', 'Votes: 0', and 'Watchers: 1 Stop watching this issue'. A red box highlights the 'Stop watching this issue' link. Below the people section is a 'Dates' section showing 'Created: 2 days ago' and 'Updated: Yesterday'.

7.1.8.3 Creating a filter rule in MS Outlook

The easiest way to filter out the content for a particular Jira issue, is creating a rule in MS Outlook which will filter out and move to a dedicated folder, those messages that include:

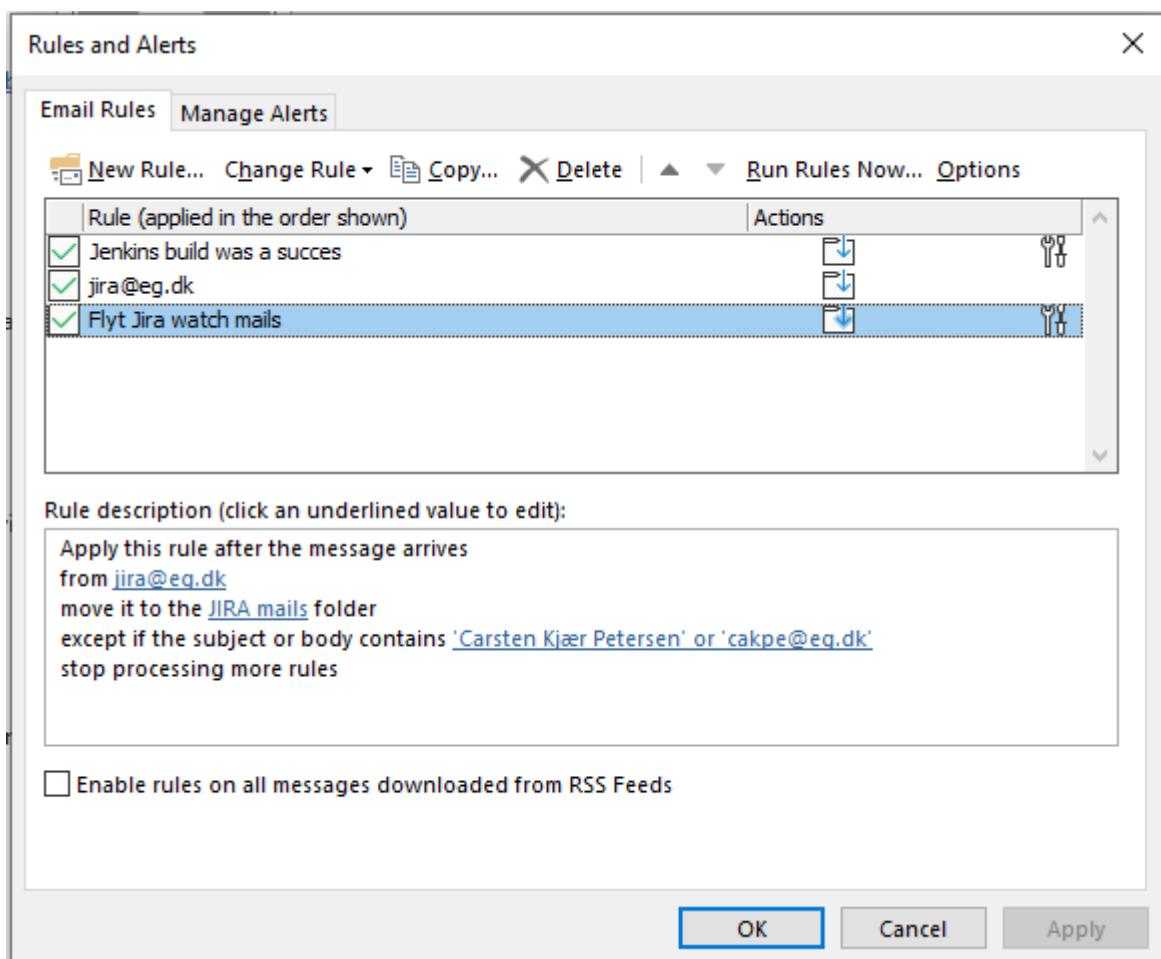
- issues where you were mentioned by another user
- issues where you were assigned as the assignee
- issues where someone has logged work for an issue you are the creator of

First of all, create a dedicated folder in your MS Outlook folder structure, to hold the notifications.

To move emails generated from events for worklog updates, you can create this rule in Outlook:

The screenshot shows the 'Rules and Alerts' dialog box in Microsoft Outlook. On the left, there's a sidebar with icons for 'Automatic Replies', 'Tools', 'Manage Rules & Alerts' (which is currently selected), 'Manage COM Add-ins', and 'Manage Add-ins'. The main area shows a list of rules under 'Email Rules'. One rule is selected: 'jira@eg.dk'. The rule description says: 'Apply this rule after the message arrives from jira@eg.dk and with worklog update in the body move it to the JIRA mails folder'. There's also an unchecked checkbox for 'Enable rules on all messages downloaded from RSS Feeds'. At the bottom, there are 'OK', 'Cancel', and 'Apply' buttons.

To move emails where your are watcher, but your name is not contained in the subject or body, you can create this rule



That is it! Your most relevant notifications will be filtered out to the folder of your choosing, allowing you to focus on things that matter.

7.1.9 7.1.9 - Creating custom HTML links to create Jira issues

Jira On-Premise provides the possibility of creating issues using HTML links, which can come in handy when referencing a colleague to create an issue quickly with predefined values for some of the parameters (fields) of the issue.

The basic URL for creating an issue with no prefilled fields is the following:

```
http://jira.atlassian.com/secure/CreateIssueDetails!init.jspa
```

In order to prepare a link with prefilled particular fields, it is sufficient to append them to the above link using a parameter key and value pair, for example:

```
http://jira.atlassian.com/secure/CreateIssueDetails!init.jspa?
param1=valueX&param2=valueY
```

where:

? - is the separator from the base URL and the parameter list; it is only used once

& - is the separator between the consecutive parameter key/value pairs

param1, param2 - are parameter keys

valueX, valueY - are parameter values

The list of parameters that can be used to prefill the issue fields can be found below:

Parameter name	Parameter key	Parameter value type	Parameter value example	Note
Project name	pid	Unique project Id	10420	please contact your agile coach / devops to acquire your project ID
Issue type	issuetype	Issue Type Id	Bug - 10004 Sub-task - 10003 Task - 10002 Story - 10001 Epic - 10000	-
Summary	summary	Plain text	Issue+created+via+Jira	-
Due date	duedate	Date	2020-10-15	-
Components	components	Component Id	10121	please contact your agile coach / devops to acquire the components' IDs
Fix version/s	fixVersions	Version Id	10121	please contact your agile coach / devops to acquire the Fix Versions' IDs
Assign To	assignee	Username	xxagz@eg.dk	-

Parameter name	Parameter key	Parameter value type	Parameter value example	Note
Reporter	reporter	Username	xxagz@eg.dk ⁵¹	-
Description	description	Plain text	Description+in+Jira	-
Epic Link	customfield_10101	Epic issue key	ONB-1516	-

The system specific IDs can be acquire by examining the page source code - there is no simple, official way to acquire that; if you need support in doing so please ask your dedicated agile coach or devops engineer, if the following tips will not help:

example: It is visible that the Project Id (pid) for project SD RM is 10700

7.1.9.1 Some examples of create issue URLs with prefilled parameters (fields):

The following URL creates a **Story** issue type in the **SDR** project with **Summary** set as *This is a user story*, **reported by** user xxdsm@eg.dk⁵² and **DueDate** set to [10 Dec 2020](#)

```
https://jira.eg.dk/secure/CreateIssueDetails!init.jspa?
pid=10700&issuetype=10001&summary=This+is+a+user+story&reporter=xxdsm@eg.dk&duedate=2020-12-10
```

The following URL creates a **Bug** issue type in the **ONB** project with **Epic Link** set as ONB-1516, **Component** set as *Onboarding issues* and **assigned** to user amsho@eg.dk⁵³

⁵¹ mailto:xxagz@eg.dk

⁵² mailto:xxdsm@eg.dk

⁵³ mailto:amsho@eg.dk

```
https://jira.eg.dk/secure/CreateIssueDetails!init.jspa?
pid=10402&issuetype=10004&customfield_10101=ONB-1516&components=10121&assignee=amsho@eg.dk
```

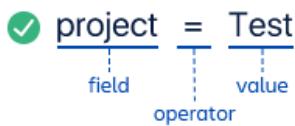
7.1.10 7.1.10 - Jira JQL hints, useful examples and managing filters

Under this chapter, you will find useful hints and JQL query examples that you can use, adjust and apply to your day-to-day work to get a specified list of issues.

JQL queries are composed of three basic parts: fields, operators, values, and keywords.

- **Fields** in JQL is a word that represents a Jira field (or a custom field that has already been defined in Jira).
- **Operators** are one or more symbols or words that compare the value of a field on its left with one or more values (or functions) on its right, such that only true results are retrieved by the clause. Some operators may use the NOT keyword. Common operators include equals (=), not equals (!=), less than (<), etc.
- **Values** are the actual data in the query. They are usually the item for which we are looking.
- **Keywords** keyword in JQL is a word or phrase that does (or is) any of the following:
 - joins two or more clauses together to form a complex JQL query
 - alters the logic of one or more clauses
 - alters the logic of operators
 - has an explicit definition in a JQL query
 - performs a specific function that alters the results of a JQL query.
- **Functions** in JQL appear as a word followed by parentheses, which may contain one or more explicit values or Jira fields. Functions are supporting specific fields and operators (please refer to each function to read about the details)

A simple query in JQL (also known as a “clause”) consists of a field, followed by an operator, followed by one or more values or functions. For example:



Under the below table, you will find the list (with reference links) of Jira fields, operators keywords, and functions.

Fields (not included NGA custom fields)	Operators	Keywords	Functions
LINK⁵⁴ (detailed description)	LINK⁵⁵ (detailed description)	LINK⁵⁶ (detailed description)	LINK⁵⁷ (detailed description)

54 <https://confluence.atlassian.com/jirasoftwareserver0814/advanced-searching-fields-reference-1043892737.html>

55 <https://confluence.atlassian.com/jirasoftwareserver0814/advanced-searching-operators-reference-1043892740.html>

56 <https://confluence.atlassian.com/jirasoftwareserver0814/advanced-searching-keywords-reference-1043892739.html>

57 <https://confluence.atlassian.com/jirasoftwareserver0814/advanced-searching-functions-reference-1043892741.html>

Fields (not included NGA custom fields)	Operators	Keywords	Functions
Affected version Approvals Assignee Attachments Category Comment Component Created Creator Custom field Customer Request Type Description Due Environment Epic link Filter Fix version Issue key Issue link type Labels Last viewed Level Original estimate Parent Priority Project Remaining estimate Reporter Request channel type Request last activity time Resolution Resolved SLA Sprint Status Summary Text Time spent Type Updated Voter Votes Watcher Watchers Work log author Work log comment Work log date Work ratio	EQUALS: = NOT EQUALS: != GREATER THAN: > GREATER THAN EQUALS: >= LESS THAN: < LESS THAN EQUALS: <=br/>IN NOT IN CONTAINS: ~ DOES NOT CONTAIN: !~ IS IS NOT WAS WAS IN WAS NOT IN WAS NOT CHANGED	AND OR NOT EMPTY NULL ORDER BY	approved() approver() breached() cascadeOption() closedSprints() completed() componentsLeadByUser() currentLogin() currentUser() earliestUnreleasedVersion() elapsed() endOfDay() endOfMonth() endOfWeek() endOfYear() everbreached() futureSprints() issueHistory() issuesWithRemoteLinksByGlobalId() lastLogin() latestReleasedVersion() linkedIssues() membersOf() myApproval() myPending() now() openSprints() paused() pending() pendingBy() projectsLeadByUser() projectsWhereUserHasPermission() projectsWhereUserHasRole() releasedVersions() remaining() running() standardIssueTypes() startOfDay() startOfMonth() startOfWeek() startOfYear() subtaskIssueTypes() unreleasedVersions() updatedBy() votedIssues() watchedIssues() withinCalendarHours()

7.1.10.1 Where JQL can be used?

JQL can be used in multiple ways:

- as ad-hoc search for specific issues;
- when saved to filter:
 - as a specific search that can be frequently used by multiple users
 - as an input for setting up the boards
 - as an input for the specific dashboard add-ons

When saving a filter please remember to set the proper permissions up so the others can use it.

7.1.10.2 Useful JQL examples

Below you will find Useful JQL queries examples that you can use, adjust or combine to fit your needs:

Use case: Issues in project that were worked on last week, but are not yet completed

(i) project = "EG Xellent" and status WAS "In Progress" DURING (startOfWeek(-1),startOfWeek())
AND statusCategory != Done

Use case: Issues that you escalated to another team a few days ago that may need a follow-up

(i) Project = Xel and assignee CHANGED FROM currentUser() BY currentUser() BEFORE
endOfDay(-3) AND resolution is EMPTY

Use case: Issues that changed specific status in the certain time period

(i) Project = Xel and status changed during (2021-03-03, 2021-04-09) from "Ready for Review" to
"Done"

Use case: Issues in the project that didn't move forward for a certain amount of time in the current sprint with specific text in the summary

- ⓘ Project = "EG Xellent" AND status = "To Do" AND updatedDate < startOfDay("-1w") and summary ~ "test" and Sprint in openSprints()

Use case: This query can be used to list the issues that should have the release notes updated. When you add this query to the Jira dashboard filter results view and you will set the fields to show "Client release notes" and "technical release notes, you will be able to see what is missing.

- ⓘ "Team name" = "Team SCM" AND status = "Ready for review" AND ("Include in client release notes" = Yes OR "Include in technical release notes" = Yes) AND status = "Ready for review"

Use case: Issues that were moved back from the Product Owner in a certain period of time backwards

- ⓘ Project = "EG Xellent" and status changed FROM "ready for review" to "In Progress" and status changed AFTER startOfDay(-7d)

Use case: Filtering the issues with missing parts (estimation, description, release notes not selected, ERP activity not assigned) for the Refinement meetings. The issue candidates are stored in future sprint slots

- ⓘ project = Zyn and sprint in futureSprints() and ("Include in client release notes" is EMPTY or "ERP Activity" is EMPTY or description is EMPTY or "Story Points" is EMPTY) and status not in (Ready, "In Progress", "Ready for review")

Use case: Query that can be used in the board configuration and will show dynamically all my issues in the current sprint no matter which team I support (great for the shared resources overview):

- ⓘ project = "EG Xellent" and assignee = currentUser() and Sprint in openSprints() ORDER BY Rank ASC

Use case: Find all issues after the creation of one specific issue:

```
(i) project = "EG Xellent" and issuekey > XEL-7190
```

Use case: Find all issues done in a specific timeframe:

```
(i) project = "EG Xellent" and resolutiondate > startOfMonth(-1) AND resolutiondate < endOfMonth(-1)
```

Use case: Find all issues that you are watching:

```
(i) project = "EG Xellent" and watcher = currentUser()
```

Use case: Find all issues which should be done before end of the week:

```
(i) project = "EG Xellent" and duedate <= endOfWeek() and resolution is EMPTY ORDER BY duedate ASC
```

7.1.10.3 Use case: for nearest sprint planning - find the issues with close (2 weeks EOW) due date.
You can of course customize the date range

```
(i) project = "EG Xellent" and duedate <= endOfWeek(2) and status != Done ORDER BY duedate ASC
```

7.1.10.4 Other resources

7.1.10.5 You can download and print or store below JQL Cheat Sheet that in a very synthetic and brief way is guiding on how to use JQL queries:



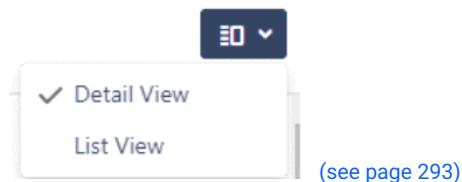
Under this link, you will find very broad guidance and overview on how to perform Jira advanced JQL search:
<https://mraddon.blog/2015/05/27/jql-reference-manual-how-to-perform-an-advanced-search/>

7.1.10.6 Exporting the JQL search results to the MS Excel Sheet for further data aggregation

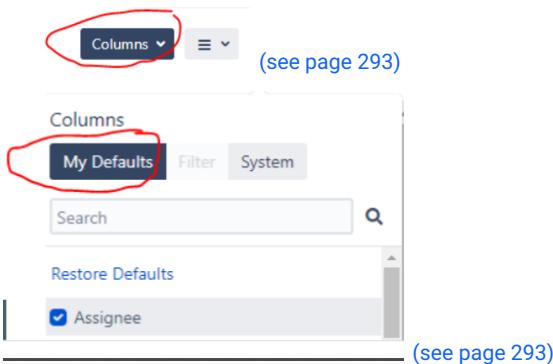
If for any reason we want to aggregate or share outside the Jira data (i.e. with customers or EG stakeholders), **there is a way to export the filter or query results to .csv file and do any required operations directly in MS Excel**. It will take some time but will broaden your data operations possibility gravely.

7.1.10.7 Exporting the filter results to the Excel sheet

Open on the right-side panel please find expand the view menu and select "list view" option:



On the list view please go to the "Columns" drop-down and "My Defaults" part to specify the data (columns) you want to export.



When you specify all please go to the export button on the top right corner and select "CSV (Current fields) from the list. Next, choose the **Semicolon** as a delimiter.

Open the exported file in MS Excel.

Done!

7.1.10.8 Jira filters

7.1.10.8.1 Creating filters in Jira

Once you've created the relevant search, you just need to save it to create a filter. All you need to do is to click "Save as" on the top menu, name it and save.

Save filter

Filter Name* CIS - close duedate items

Enter a name for this Filter

Save Cancel

7.1.10.8.2 Sharing a Filter in Jira

Having created your filter, you can now share it with whoever you need by customizing its sharing settings. As well as saving on doubling-up work, this means the system won't become overburdened with replicas of the same filters. To do this:

- Go to your filter > Choose Details > Edit permissions

Or

- Go to the Filters dropdown > select the “...” button for the filter you want to share > Edit



In this new window, you can edit details such as the filter name, the filter description, and favorites settings. You can also edit the sharing settings.

Setting up filter shares

Sharing a filter is very important. If you have prepared the filter for a group of people that will use it (i.e. in Jira dashboards), you need to add them (project, group or any logged user) to the filter shares. **Otherwise, they will not be able to see the results of the filter** if it remains private or not shared properly.

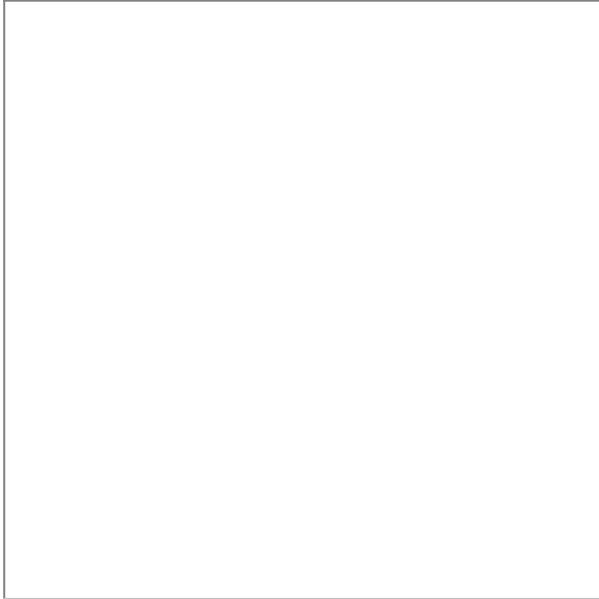
7.1.10.8.3 Subscribing to Jira Filters

You and your teammates can also subscribe to filters so you get regular notifications of the updated results for a specific search query. This could be used to flag critical issues that haven't been resolved, for example, or to keep track of all the open issues assigned to you, which could include anything from "All resolved issues for a particular project every week" to "All issues managed for invoicing at the end of every month".

7.1.10.8.3.1 To get an email subscription to a filter

- Open your filter and click "Details"

- Then click “New subscription”.



This new configuration window allows you to define how you want to manage the subscription:

- Who is linked to the subscription: personal or shared via groups (only including groups of which you are a member)
- Frequency of delivery: daily, for a specific day of the week, for a specific day of the month, or advanced
- Interval of delivery:
 - Daily: once a day, every x hours, etc.
 - Weekly: once a day, every x hours, etc. on a given day
 - Monthly: selection of a particular day and time
 - Advanced: you can configure with a cron expression
- You can also decide whether you want to receive an email even if the filter does not have any results

For instance, you could choose to receive an alert two days a week at 8 am and to receive an update even if the filter doesn't have any results. Do note, though, that the subscription will only deliver the first 200 results from the filter.

The subscription system is extremely useful because it enables a scheduled system of updates. But if the result of the filter changes after the notification is sent, then you won't be informed of these changes.

Because of this, dashboard widgets are generally preferred. They are more dynamic, as they display content extracted directly from the tool.

7.1.10.8.4 Where you can use the filters in Jira?

Jira filters can be used in many places in the system and for multiple purposes, such as:

- **Issue Navigator:** The primary and most common place where filters are used is the Issue Navigator. It allows you to create, save, and manage filters to search for and display specific issues that match

the criteria you define. You can use filters to view issues assigned to you, issues with specific labels, issues in specific projects, and much more.

- **Dashboards:** Filters can be used to populate the gadgets on your Jira dashboards. For example, the "Filter Results" gadget allows you to display the list of issues returned by a particular filter on a dashboard.
- **Reports:** Filters can be utilized to generate custom reports and track progress on specific sets of issues. In the "Issue Statistics" and "Created vs. Resolved Chart," you can apply a filter to display only the issues you are interested in analyzing.
- **Jira Automation:** Filters are also used in Jira Automation to define the scope of automation rules. For instance, you can set up an automation rule to notify a specific group of people whenever an issue that matches a particular filter is created or updated.
- **Agile Boards:** If you are using Jira's Scrum boards or Kanban boards, and you have proper permissions, you can use filters to determine which issues are displayed on those boards. By setting up board filters, you can decide which issues should be visible and managed on a particular board.
- **Project-specific views:** In some configurations, filters can be used to customize the issue views within specific projects. This can help project administrators tailor the issue view to their teams' specific needs.

7.1.11 7.1.11 - Jira common boards setup for cross-product collaboration

Sometimes the teams from different Business Units or supporting different products between Jira projects are joining forces to produce a common solution. In such cases sometimes the Jira business project view is not enough, because we can view and manage issues in a limited overview. More about business projects you can read [HERE⁵⁸](#).

To have a full standard overview of the common work delivery, dedicated boards in each Jira project (product) can be created. The instruction on how to create common boards is available below.

7.1.11.1 Setting up common Jira board

7.1.11.1.1 Distinguish the common area in each Jira project

The scope of the common work needs to be marked in each Jira project. We have to do it to be able to prepare a dedicated common board JQL filter. The distinction can be made in multiple ways:

- **Use the specific Epics** to group the scope of the common work (there can be more than one epic). The epic naming convention can be different in each project. Even if we use the same Epic summary in each project, the Epic key will be different.
- **Use specific components** and mark all the issues with these components to distinguish the common scope of work. The components have to be created separately in each Jira project and they don't need to have a common naming convention.
- **Use a dedicated label/s.** In this scenario, you can use a common (the same) label/s, because the labels are available across Jira projects. You need to be careful with marking all the issues because

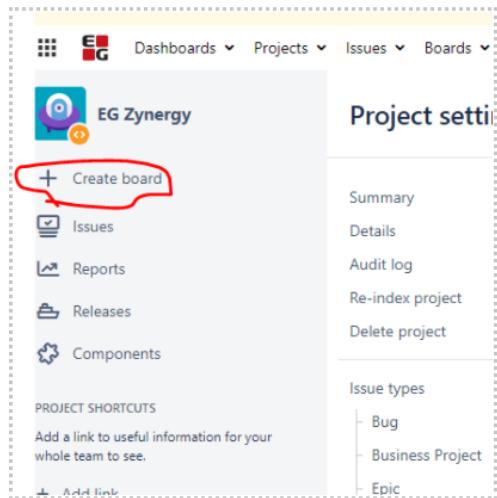
⁵⁸ <https://community.atlassian.com/t5/Jira-Work-Management-Articles/What-is-a-business-project/ba-p/1693132>

labels can be created by anybody and any typo in the label naming can result in a new label creation and finally not proper issue marking.

7.1.11.1.2 Create a dedicated board in one Jira project

The board should have the same setup as both teams use. In the case of a different estimation technique for each team, the teams need to align the approach here. There is no possibility to create a common board for the teams that are using a different framework (Jira and Kanban). The board should have a default EG Setup. In case of any differences, the board creation should be consulted with the NGA Agile Coach.

Please go to your project in Jira and in the left sidebar menu please select "+ Create board":



From the available boards please select "Create a Scrum board":

Create a board

Scrum

Scrum focuses on planning, committing and delivering time-boxed chunks of work called Sprints

[Create a Scrum board](#)

[Create a Scrum board with sample data](#)

Kanban

Kanban focuses on visualising your workflow and limiting work-in-progress to facilitate incremental improvements to your existing process

[Create a Kanban board](#)

[Create a Kanban board with sample data](#)

[Cancel](#)

From the below list please select "Board from an existing project" and select "Next" (if you already have your board JQL filter ready, you can choose the 3rd option "Board from an existing Saved Filter" and provide the filter in the following step):

Create a board

- Board created with new Software project

A new board based on a new Software project

- Board from an existing project

Boards can contain one or more projects.

- Board from an existing Saved Filter

An advanced option using a JQL filter.

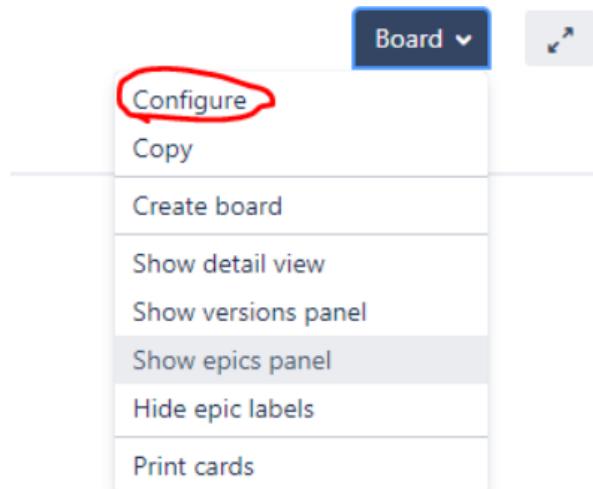
[Back](#)

[Next](#)

Please provide your board name. After it please select: "Create board" function.

Configure your newly created board

After creating the board from the board perspective, in the upper right corner please select "Board" and from the drop-down: "Configure":



In the left sidebar at settings space please select: "Columns":

In Columns please do the following actions:

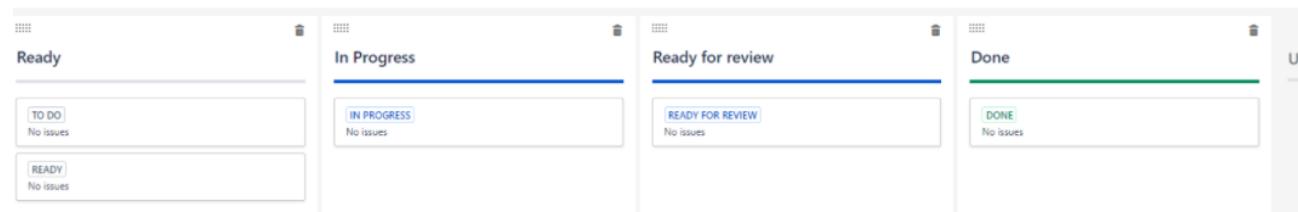
- rename the "To Do" column to: "Ready";
- add one additional column by choosing "Add column" on the right side and name it: "Ready for review"
- please put the right order of the columns by moving them left or right to have the following order (from left side): "Ready" "In Progress" "Ready for Review" and "Done"
- next please add the right statuses by moving them under particular columns in the following order of mapping:
 - "Ready" -> "TO DO" and "READY"
 - "In Progress" -> "IN PROGRESS"
 - "Ready for Review" -> "READY FOR REVIEW"
 - "Done" -> "DONE"

After doing all the above actions you should get the following setup for columns:

Column management

Columns can be added, removed, reordered and renamed. Columns are based upon global statuses and can be moved between columns. Minimum and maximum constraints can be set for each mapped column.

Column	Status Mappings
Ready	TO DO READY
In Progress	IN PROGRESS
Ready for review	READY FOR REVIEW
Done	DONE



Next step is proper adding Card colors. To do it please choose the "Card colors" from the settings left sidebar and from the expand list please select "Assignees":

Settings for EG Xena Board

SETTINGS

- General
- Columns
- Swimlanes
- Quick Filters
- Card colors**
- Card layout
- Estimation
- Working days
- Issue Detail View

Card colors

Choose a method for assigning colors to your cards. If no method is selected, Jira will save immediately, so you can switch back to it later if you wish.

[Learn more about card colors.](#)

Colors based on:

Assignees

Color

If you already know what estimation technique will be used by your teams in the projects you can select the proper value in the "Estimation" part in board settings left sidebar. You can select: "Original Time Estimate" or "Story Points" and in "Time tracking" part you select: "Remaining Estimate and Time Spent". If you don't know yet this can be done later:

SETTINGS

- General
- Columns
- Swimlanes
- Quick Filters
- Card colors
- Card layout
- Estimation**
- Working days
- Issue Detail View

Estimation

Issues can be estimated when in the Backlog to get an idea of how much work needs to be done.

Estimation Statistic:

Story Points

Estimate issues in the Backlog by entering values for **Story Points**. Your velocity from sprint estimates.

Time Tracking:

None

Issues will burn down their **Story Points** value upon completion.

Remaining Estimate and Time Spent

Track time against issues using Jira's **Remaining Estimate** and **Time Spent** fields.

As soon as the board has been created and properly configured, we need to specify the board filter query, to trim the common scope from all the projects.

General and filter

The Board filter determines which issues appear on the board. It can be based on one or more projects, or custom JQL queries.

General

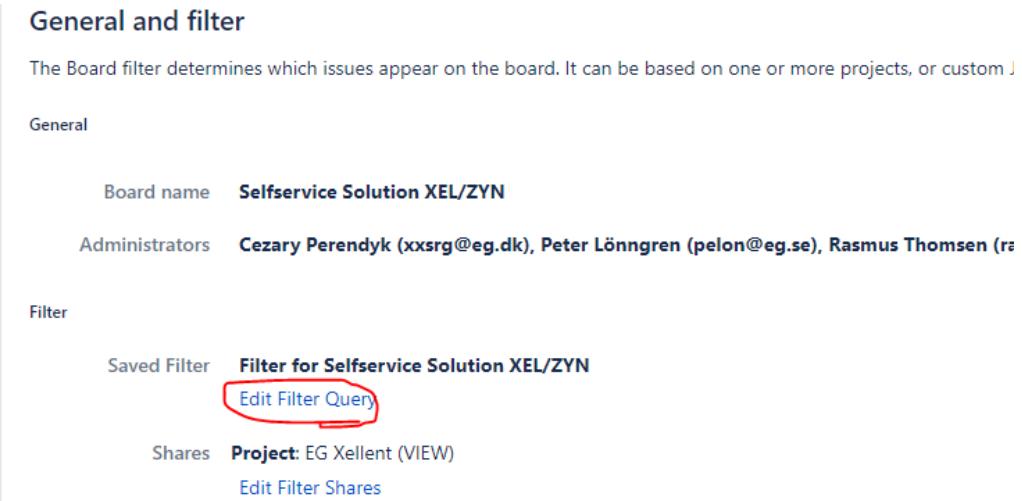
Board name **Selfservice Solution XEL/ZYN**

Administrators **Cezary Perendyk (xxsrg@eg.dk), Peter Lönngren (pelon@eg.se), Rasmus Thomsen (rasmus.thomsen@eg.dk)**

Filter

Saved Filter **Filter for Selfservice Solution XEL/ZYN**
[Edit Filter Query](#) (circled in red)

Shares **Project: EG Xellent (VIEW)**
[Edit Filter Shares](#)



Below you can see the custom board query that is specifying the common work scope:

(Project = "EG Xellent" and "Epic Link" = XEL-9110) or (project = "EG Zynergy" AND labels = ZCS AND status != done)

The above example is grouping the issues from two different projects EG Xellent and EG Zynergy. In Xellent we are marking all the issues grouped under a dedicated Epic. On the opposite in Zynergy, we are distinguishing all issues that are marked with a specific label, and the status of these issues is not done. This is a good example of the combined markers in each project.

You can combine more than one Jira project this way to reflect your current situation.

7.1.11.1.3 Add the permissions to the engaged team members across Projects

The dedicated board creation is not enough to have an overview of the issues in another Jira project. To make it possible we need to give the proper permissions and roles across the projects to be able to view all the issues. So the engaged team members in Project X should get the permissions in Project Y. At the same time, team members from Project Y need to have the permissions granted to Project X. When properly done, the common board should appear in every Jira project.

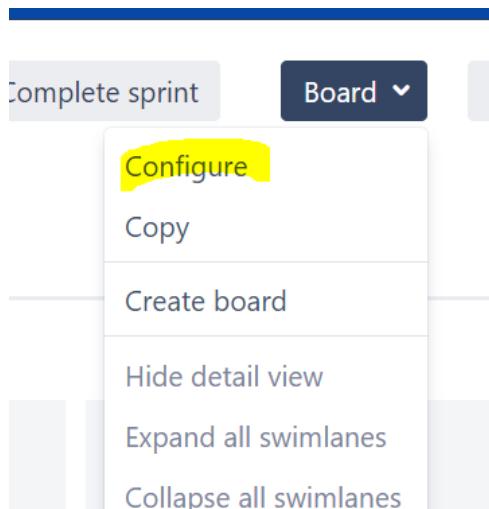
At the end please remember to establish Board Administrators to have the possibility to manage the board.

7.1.11.1.4 The final setting up

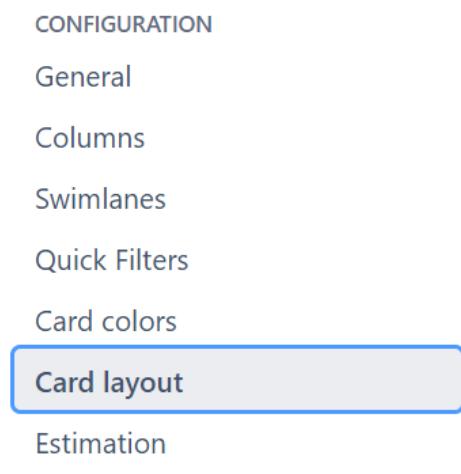
Team coverage

To have a better overview of the issues and team coverage you need to adjust the board in the following way.

Go to the board and select "Configure":



In the configuration left side menu select "Card layout":



and add to the layout "Team name" in the Backlog and Active Sprint view. This will allow you to see which team has been selected to the particular issues, so you can have an overview of the team coverage.

Project coverage

If you want to have an overview under which Jira Project the issues are being covered you can adjust it as well in the board setup. In the board configuration mode go to the "Swimlanes":

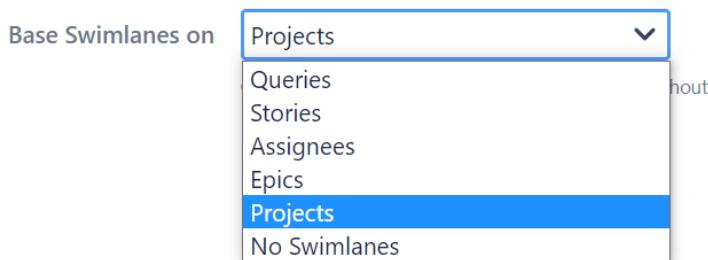
CONFIGURATION

[General](#)[Columns](#)**Swimlanes**[Quick Filters](#)[Card colors](#)[Card layout](#)

and from the drop-down select "Projects":

Swimlanes

A swimlane is a row on the board that can be used to group issues. It will not be lost when changing to another swimlane type.



If needed, you can add dedicated quick filters to support the proper board management.

7.1.11.2 Using common Jira board

The above-mentioned Jira board can be used in multiple ways to support the events and day-to-day work. Of course, the board usage is not limited to the below examples.

Use Case 1 - using the board to order the backlog items priorities in general

Having all the common work scope in a single place is giving us a great opportunity to order all the issues top-down so the priorities are clear to all engaged in the endeavor. This is being done as a combined effort by all the engaged Product Owners. If needed the Product Owners can consult the proper team representatives if technical or domain knowledge is required. This can give great input for the events and meetings that will follow.

Use Case 2 - using the board in the Cross-team pre-planning event

Since the common work is initially ordered by Product Owners the board can be used in the Cross-team pre-planning event that is being conducted by Product Owners and all teams representatives collectively. During this event, the needed details are being added to the issues including dependencies. Initial team distribution is as well being made here. The board can give us a great overview if the distribution has been made properly, the workload for the upcoming sprint is well balanced and the issues delivery sequence is properly reflected.

After the issues are distributed across the teams, the detailed refinement process can be made at the team level with the proper Product Owner or Product Owners.

Use Case 3 - using the board in the Scrum of Scrums event

The board can be successfully used at the Scrum of Scrums event where the cross-team calibration is being held in the run since we have all the scope in a single place. The board can be used similarly to the team board on a team level during Daily Scrum.

Use Case 4 - tracking day-to-day progress and dependencies

During the Sprint each team engaged in the development process, as well as Product Owners, can track the current progress. This is especially useful when there are cross-team dependencies and on a team level, we are waiting for the other team input to move forward with our issues. If needed the remaining work can be adjusted to improve the delivery process efficiency. Using marking the impediments with Flag function can substantially improve the progress visibility and the proper persons can focus their energy on impediments addressing or removal.

7.1.12 7.1.12 - Jira Automation

7.1.12.1 Jira Software Automation - overview and benefits

Jira Software Automation empowers users to streamline workflows, save time, enhance collaboration, and improve productivity. By automating routine tasks, teams can focus on more valuable work, reduce errors, and deliver projects more efficiently.

(i) Imagine the situation where you are having a routine activity in Jira and you need to create a standard ticket with the defined sub-tasks with some input pre-defined on a scheduled manner. You need to remember about it, you need to create the item manually or clone from the previous items and you need to fill in the content. With Jira automation, all of that can happen without your touch. All you need to do is create the automation and enjoy the time and energy saved as well as the complexity mitigated. Jira automations can do much more...

Jira Software Automation is the functionality that can help you with:

- **Time-saving Efficiency:** Jira Software Automation enables users to automate routine and time-consuming tasks, freeing up valuable time and resources. By automating manual processes such as issue creation, assignment, and status updates, users can focus on more strategic and value-added activities.
- **Enhanced Productivity:** With automation, teams can reduce manual errors and increase productivity. Automation rules ensure that tasks are executed consistently and accurately, minimizing the risk of human mistakes. This allows teams to work more efficiently and deliver projects faster.
- **Improved Collaboration:** Jira Software Automation promotes collaboration by automating the flow of information and updates across teams. It can automatically notify relevant stakeholders, assign

tasks, and trigger actions based on predefined rules. This streamlines communication, reduces delays, and facilitates smoother teamwork.

- **Proactive Issue Management:** Automation helps proactively manage issues by triggering actions based on predefined conditions. For example, rules can be set up to escalate overdue tasks, flag critical issues, or notify stakeholders when certain conditions are met. This proactive approach helps teams stay on top of their work and address potential problems before they escalate.

One of the significant advantages of Jira Software Automation is that it **doesn't require coding expertise**. Users can create automation rules and workflows without the need for extensive programming knowledge. To some extent, you need to have knowledge about the JQL language to be able to create the queries that will be used in the automation. This empowers users from various backgrounds to automate tasks and processes within Jira easily, making automation accessible to all users, regardless of their coding abilities.

To be able to create a Jira Automation Rule on your project level, you **need to have project administrator permissions** granted.

7.1.12.2 Understanding the automation rules and their components

There are three main components (ingredients) of Jira automation rules:

1. **Triggers:** Triggers are the events or conditions that initiate the automation rule. They define when the automation should be executed. Jira provides a wide range of triggers that can be used, such as issue created, issue updated, status transitioned, comment added, field value changed, and many more. You can select one or multiple triggers based on your requirements.
2. **Conditions:** Conditions are used to define criteria that must be met for the automation rule to be executed. They allow you to add additional logic and filtering to control when the rule should run. Conditions can be based on various factors, including issue attributes (e.g., issue type, priority), field values, issue links, dates, and more. By defining conditions, you can ensure that the automation is triggered only for the desired set of circumstances.
3. **Actions:** Actions are the operations or tasks that are performed when the automation rule is triggered and the conditions are met. They define the desired outcome of the automation. Jira offers a wide range of actions that can be performed, such as updating issue fields, transitioning issues to different statuses, sending notifications, creating subtasks, adding comments, linking issues, and more. You can select one or multiple actions to be executed as part of the automation rule.

By combining triggers, conditions, and actions, you can create powerful automation rules that automate repetitive tasks, enforce business processes, improve productivity, and enhance collaboration within your Jira instance. It's important to carefully define and configure these components to ensure that your automation rules operate effectively and deliver the intended results. In the next chapter below, you will find the reference materials explaining the components of Jira automation rules.

7.1.12.3 Getting started with Jira Automation

This chapter is not dedicated to providing you with a comprehensive introduction, overview, and training on how to create and manage Jira automation. Since the official Atlassian documentation is very good for this purpose, we will just provide you with a below source reference point:

[Jira automation introduction by Atlassian⁵⁹](#)

[Automate your project | Jira Software Data Center and Server 9.4 | Atlassian Documentation⁶⁰](#)

[Jira Manual for automation feature⁶¹](#)



Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

<https://www.youtube.com/watch?v=CK2mcvumUvl>

7.1.12.4 Example use cases for Jira Automation

Below you can find a couple of examples of Jira Automation usage. This of course doesn't fulfill the vast possibilities of this tool:

- **Issue Assignment:** Automatically assign new issues to the appropriate team member based on predefined criteria, such as issue type, component, or priority. [Reference.⁶²](#)
- **Status Updates and Transitions:** Automatically transition issues to different statuses based on specific triggers, such as a certain field value change or a time-based condition. [Reference.⁶³](#)
- **SLA Tracking and Reminders:** Set up automation to track and enforce Service Level Agreements (SLAs) by sending reminders when approaching or breaching SLA thresholds. [Reference.⁶⁴](#)
- **Notification and Commenting:** Automatically notify relevant stakeholders or team members when specific events occur, such as a new issue creation or a critical issue update. You can also automate adding comments to an issue based on certain conditions. [Reference.⁶⁵](#)
- **Subtask Creation:** Automatically generate subtasks for new or specific types of issues, helping break down larger tasks into manageable subtasks and assigning them to the appropriate individuals. [Reference.⁶⁶](#)
- **Issue Linking:** Automate the linking of related issues based on predefined conditions or triggers, creating connections between issues for better visibility and traceability. [Reference.⁶⁷](#)
- **Issue Labeling and Categorization:** Automatically add labels or categorize issues based on specific criteria, making it easier to filter, search, and report on related issues. [Reference⁶⁸](#)
- **Issue Prioritization:** Automate the prioritization of issues based on predefined criteria, such as impact, urgency, customer, creator, and source (i.e. ServiceNow). This can help ensure that high-priority issues are addressed promptly. [Reference.⁶⁹](#)

⁵⁹ <https://www.atlassian.com/software/jira/guides/automation/overview#what-is-automation>

⁶⁰ <https://confluence.atlassian.com/jirasoftwareserver0904/automate-your-project-1188765970.html>

⁶¹ <https://confluence.atlassian.com/automation/jira-automation-data-center-and-server-993924595.html>

⁶² <https://www.atlassian.com/agile/tutorials/how-to-automatically-assign-issues-with-jira-software-automation>

⁶³ <https://www.atlassian.com/devops/automation-tutorials/jira-automation-rule-to-transition-issues>

⁶⁴ <https://community.atlassian.com/t5/Marketplace-Apps-Integrations/How-to-set-a-reminder-in-JIRA-with-the-Automation-for-JIRA-tool/qaq-p/1439395>

⁶⁵ <https://www.youtube.com/watch?v=ccWTlgdnQqE>

⁶⁶ <https://www.atlassian.com/agile/tutorials/how-to-auto-create-subtasks-with-jira-software-automation>

⁶⁷ <https://confluence.atlassian.com/automationkb/automatically-link-issues-to-other-issues-with-automation-for-jira-1130727671.html>

⁶⁸ <https://community.atlassian.com/t5/Jira-questions/Automation-for-Jira-Automatically-Add-Label-to-Stalled-Tickets/qaq-p/1296772>

⁶⁹ <https://support.atlassian.com/jira-service-management-cloud/docs/create-an-automation-rule-to-prioritize-your-service-desks-incident/>

- **Issue template creation:** If you want to keep the standardized template for a story, bug, or task, you can pre-define the Jira item description with your template. You can as well pre-define some of the Jira fields based on your specified conditions. [Reference⁷⁰](#).

Under [this link⁷¹](#), you will find the repository for the most common and popular Jira automation rules. Remember, Jira automation is highly customizable, and these examples can be adapted to fit your specific requirements.

7.1.12.5 Where can I practice Jira automation? Where to get support if needed?

Atlassian is offering a free playground sandbox environment where you can interactively explore 100s of automation templates to see exactly how they work. You can find it under [this link⁷²](#) (bottom of the linked page).

If you are not proficient enough to prepare the suitable Jira Automation Rule, or you struggle with it - you can always reach the **NGA team** and ask for help.

7.1.12.6 I have a proven Jira automation rule that could be used in other projects across EG - How do I share it?

Jira is allowing you to set and use the Jira Automation rules at your project level. If you have proven Jira automation that could be used by other Business Units and Jira Projects efficiently, you may ask the NGA team to create a global template rule and add it to the templates repository. From the repository, it can be duplicated into the specific project, and adjusted to the local needs. Please reach NGA team with such requests. Please review the below screenshot for reference:

Name	Owner	Project	Enabled
alystra_subtask	Tomasz Wiktorowicz (ADM) [X]	Global	<input checked="" type="checkbox"/>
backup_activity	Tomasz Wiktorowicz [X]	Global	<input type="checkbox"/>
copy_team_name_from_parent	Tomasz Wiktorowicz [X]	Global	<input type="checkbox"/>
Create subtasks Landax	Lukasz Monecki (ADM)	Global	<input checked="" type="checkbox"/>

7.1.12.7 Good practices for creating and managing Jira Automation Rules

Below you will find hints and good practices that will help you in good and efficient Jira Automation Rules creation and maintenance:

⁷⁰ <https://community.atlassian.com/t5/Jira-questions/How-can-I-create-an-User-Story-template-in-Jira/qaq-p/1011870>

⁷¹ <https://www.atlassian.com/software/jira/automation-template-library#/rule-list?systemLabelId=all&page=1&pageSize=20&sortKey=name&sortOrder=ASC>

⁷² <https://www.atlassian.com/software/jira/automation-template-library#/rule-list?systemLabelId=all&page=1&pageSize=20&sortKey=name&sortOrder=ASC>

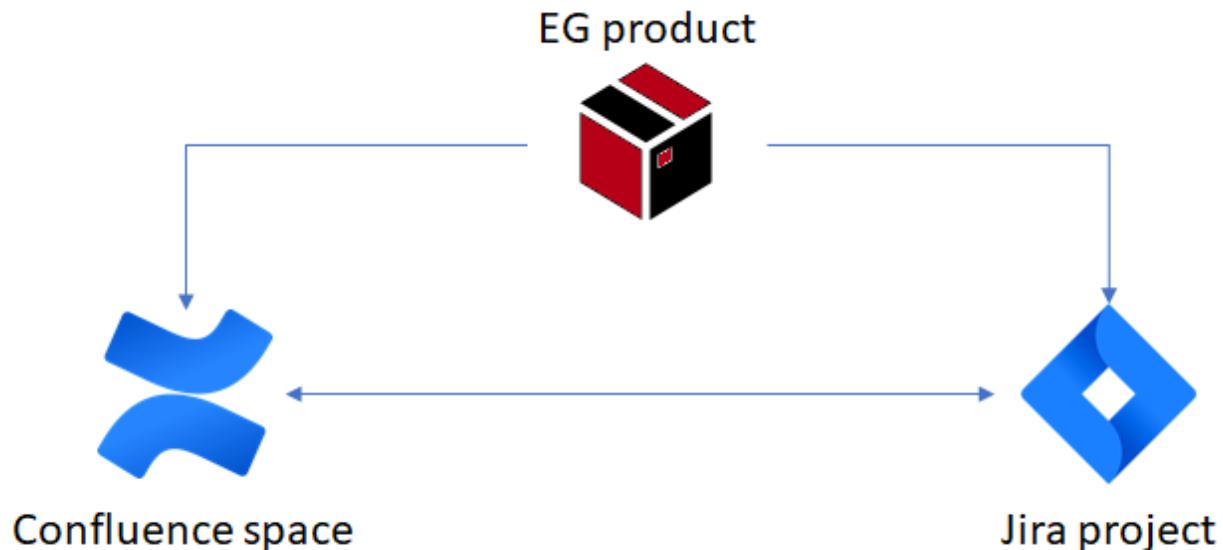
1. **Start Small and Iterate:** Begin with simple automation rules and gradually expand as you gain more experience. Start with automating repetitive and time-consuming tasks before moving on to more complex processes. Iterate and refine your rules based on feedback and evolving requirements.
2. **Clearly Define Automation Goals:** Clearly define the goals and objectives you want to achieve through automation. Identify specific pain points, bottlenecks, or inefficiencies that can be addressed through automation. Having a clear vision helps you design focused and effective automation rules.
3. **Analyze and Map Existing Workflows:** Before implementing automation, thoroughly understand your existing workflows and processes. Identify the key steps, decision points, and triggers that can be automated. This analysis ensures that your automation aligns with your actual workflow and doesn't introduce conflicts or redundancies.
4. **Maintain Simplicity and Clarity:** Keep your automation rules simple and easy to understand. Avoid unnecessary complexity or over-engineering. Ensure that the rules are clear, concise, and well-documented so that they can be easily maintained and understood by both current and future team members.
5. **Test and Validate:** Before deploying automation rules in a production environment, thoroughly test and validate them. Use a test environment to ensure that the rules function as expected and don't have any unintended consequences. Test different scenarios and edge cases to verify the reliability and accuracy of the automation.
6. **Monitor and Review:** Regularly monitor and review the performance and effectiveness of your automation rules. Keep an eye on the execution, trigger conditions, and outcomes of automated actions. This helps identify any issues, fine-tune the rules, and ensure they continue to deliver the desired results. This can be done by reviewing the "audit log" inside the automation details.
7. **Communicate and Involve Stakeholders:** Clearly communicate the purpose, benefits, and potential impacts of automation to all relevant stakeholders. Involve the relevant teams, including administrators, project managers, and end-users, in the design and implementation process. This ensures that automation aligns with the needs and expectations of all involved parties.
8. **Consider Maintenance and Upgrades:** Automation rules may require updates or modifications over time. Plan for maintenance and upgrades as your workflows or requirements evolve. Regularly review and optimize your automation rules to ensure they continue to meet your changing needs.
9. **Balance Automation and Manual Processes:** While automation can bring numerous benefits, it's essential to strike a balance between automation and manual processes. Evaluate the tasks and processes that genuinely require automation versus those that may be better suited for manual handling. Consider the human factor and the need for flexibility and judgment in certain scenarios.

7.2 7.2 - Confluence

- [7.2.1 - Confluence space structure \(see page 319\)](#)
- [7.2.2 - Documenting a Sprint Retrospective \(see page 321\)](#)

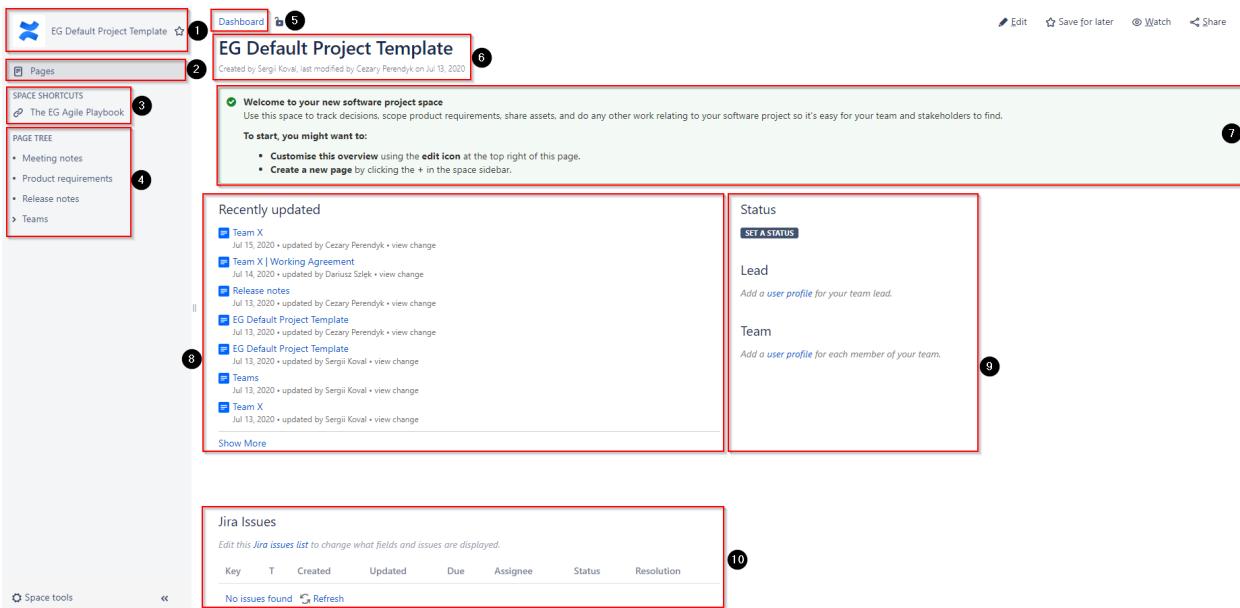
7.2.1 7.2.1 - Confluence space structure

The following is a default proposed product space to be used for all products across EG as a Confluence space representation. The space corresponds to one Jira project and uniquely identifies a product as defined for EG.

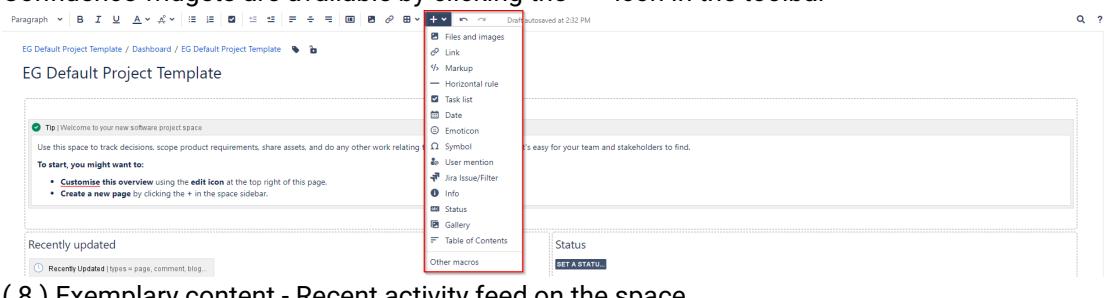


The default space uses a [software project](#) Confluence template with several elements:

- an Overview page / space dashboard - can contain general product information, status and information about the teams, components, etc.
- a tree-like hierarchy page structure, customizable to some extent by the Product Owner, consisting of a couple of predefined pages, such as:
 - Meeting notes - for registering meeting notes and outcomes
 - Product requirements - main part of the space, for registering and maintaining product requirements
 - Release notes - for generating, collecting and distributing version or filter-based release notes from the resulting list of resolved issues with their client/technical noted comments
 - Teams - a page serving as a catalogue and combining in one place all team pages working in scope of the product; the underlying team pages contain team specific information, such as: team backlogs, team member list, team events schedule, team impediments, team maturity, team metrics etc.
 - Retrospectives - for creating and reviewing team retrospective notes, resulting from conducted team Sprint Retrospective events



1. Confluence space - product - name (also a link to the main overview page of the space)
2. Link to the space dashboard with pages and page activity updates
3. Customizable section for defining jira/confluence page shortucts
4. Tree-like page structure for the product / confluence space
5. Link to the space dashboard with pages and page activity updates
6. Confluence space - product - name, including creation and modification dates and author
7. Intro gadget (presented by default, removable)
8. 9. 10. Overview page customizable space - to be filled with content, confluence widgets
 - a. Confluence widgets are available by clicking the "+" icon in the toolbar



9. Technical setup

This section should contain all the links that are relevant for the project:

- ▼ Technical setup
 - Source code
- ▼ Builds
 - Worker nodes

a. The **Source code** page should contain the list of links to the source code repositories:

Next Generation Agile / The tooling / Technical setup 🔒

Source code



Created by Jaroslaw Krochmalski
Just a moment ago • 🔖 Analytics

[eg_as/eg-housing](#)

[eg_as/eg-housing_temp](#)

b. The **Builds** page should contain the list of links to jobs on Jenkins, with comments if needed:

Next Generation Agile / The tooling / Technical setup 🔒

Builds



Created by Jaroslaw Krochmalski
Just a moment ago • 🔖 Analytics

<https://jenkins.eg.dk/view/EG-Housing/>

https://jenkins.eg.dk/view/EG-Housing/job/eg-housing_D_V3_F/

https://jenkins.eg.dk/view/EG-Housing/job/eg-housing_D_V3_B/

c. the list of Jenkins worker nodes (with descriptions of the technology stack and tools installed).

7.2.2 - Documenting a Sprint Retrospective

After a successful Sprint Retrospective, it is good to document it for future inspirations, backwards inspection and improvements tracking. Your confluence space should contain a page called *Team <name> / Retrospectives* - this is where the notes should be stored.

The screenshot shows a Jira Confluence page for 'Team X | Retrospectives'. The left sidebar includes 'EG Default Project Template' with 'Pages', 'SPACE SHORTCUTS' (The EG Agile Playbook), 'Retrospectives', 'PAGE TREE' (Meeting notes, Product requirements, Release notes, Retrospectives), and 'Teams' (Team X). Under 'Team X', there is a section for 'Team X | Retrospectives' containing a list of retrospective notes from July 2020.

Incomplete Tasks:

Description	Due date	Assignee	Task appears on
<input type="checkbox"/> <input checked="" type="checkbox"/> TRT-14 - [RETROS] Implement one final suggestion to make our process better TO DO			2020-07-02 Sprint 5 Retrospective

All retrospective notes:

Title	Date	Participants
2020-07-02 Sprint 5 Retrospective	02 Jul 2020	@Dariusz Szék, @Jarosław Krochmaliski, @Cezary Perendyk, @Łukasz Halicki
2020-07-09 Sprint 6 Retrospective	09 Jul 2020	@Dariusz Szék, @Łukasz Halicki, @Cezary Perendyk, @Jarosław Krochmaliski, @Tomasz Wiktorowicz
2020-07-16 Sprint 7 Retrospective	16 Jul 2020	@Dariusz Szék, @Łukasz Halicki, @Cezary Perendyk, @Tomasz Wiktorowicz
2020-07-23 Sprint 8 Retrospective	23 Jul 2020	@Dariusz Szék, @Jarosław Krochmaliski, @Tomasz Wiktorowicz

Catalog of retrospective notes:

- 2020-07-02 Sprint 5 Retrospective
- 2020-07-09 Sprint 6 Retrospective
- 2020-07-16 Sprint 7 Retrospective
- 2020-07-23 Sprint 8 Retrospective

Annotations on the page:

- ① A red box highlights the 'Create retrospective' button in the top right corner.
- ② A red box highlights the list of incomplete tasks.
- ③ A red box highlights the list of retrospective notes.
- ④ A red box highlights the catalog of retrospective notes in the 'Teams' section of the sidebar.

By entering this page, the following elements should be visible:

- *Create retrospective* button for creating a page template for a Sprint Retrospective (1)
- a list of incomplete tasks (improvements) mentioned in the all of the retrospective notes (2)
- a list of all retrospective notes, including their dates and participants (3)
- a catalog of all retrospective notes in the space catalogue structure (4)

7.2.2.1 Creating a Sprint Retrospective note

After pressing the *Create retrospective note* button, the following page should be created:

Dashboard / ... / Team X | Retrospectives Edit Save for later Watching Share

2020-07-02 Sprint 5 Retrospective

Created by Dariusz Ślięć, last modified 7 minutes ago

Date	02 Jul 2020
Participants	@Dariusz Ślięć @Jarosław Krocinski @Cezary Perendyk @Lukasz Halicki

Retrospective

Sprint 5 retrospective Set the context of the retrospective here...

Liked	Learned	Lacked	Longed For	Action points
<input checked="" type="checkbox"/> We plan our work using akanban board and we have more time for planning	<input checked="" type="checkbox"/> We can use kanban to manage our tasks and we have more time for planning	<input checked="" type="checkbox"/> We lack communication with our staff	<input checked="" type="checkbox"/> We lack communication with our clients, planning meeting of clients	<input checked="" type="checkbox"/> Create regular 10 minute meeting for the development group
<input checked="" type="checkbox"/> team understanding everything in code	<input checked="" type="checkbox"/> What stage of business will be developed	<input checked="" type="checkbox"/> We lack communication with our clients	<input checked="" type="checkbox"/> We lack communication with our clients, planning meeting of clients	<input checked="" type="checkbox"/> Create more communication meetings between clients
<input checked="" type="checkbox"/> we will have good communication in the future	<input checked="" type="checkbox"/> Good way of working on project	<input checked="" type="checkbox"/> We lack communication with our clients	<input checked="" type="checkbox"/> We lack communication with our clients, planning meeting of clients	<input checked="" type="checkbox"/> Regular 10 minute meeting for the development group
<input checked="" type="checkbox"/> Our team can handle complex problems	<input checked="" type="checkbox"/> Encourage people to talk to each other	<input checked="" type="checkbox"/> We lack communication with our clients	<input checked="" type="checkbox"/> We lack communication with our clients, planning meeting of clients	<input checked="" type="checkbox"/> Create more communication meetings between clients
<input checked="" type="checkbox"/> We have very fast feedback loop (we receive feedback from our clients)	<input checked="" type="checkbox"/> Encourage people to talk to each other	<input checked="" type="checkbox"/> We lack communication with our clients	<input checked="" type="checkbox"/> We lack communication with our clients, planning meeting of clients	<input checked="" type="checkbox"/> Regular 10 minute meeting for the development group
<input checked="" type="checkbox"/> Our team has a lot of experience	<input checked="" type="checkbox"/> Encourage people to talk to each other	<input checked="" type="checkbox"/> We lack communication with our clients	<input checked="" type="checkbox"/> We lack communication with our clients, planning meeting of clients	<input checked="" type="checkbox"/> Create more communication meetings between clients
<input checked="" type="checkbox"/> We have a lot of experience	<input checked="" type="checkbox"/> Encourage people to talk to each other	<input checked="" type="checkbox"/> We lack communication with our clients	<input checked="" type="checkbox"/> We lack communication with our clients, planning meeting of clients	<input checked="" type="checkbox"/> Regular 10 minute meeting for the development group

Actions & Decisions

- FRT-12 - [RETROS] Do some improvement for the Team DONE
- FRT-13 - [RETROS] Do another improvement on the tooling side DONE
- TRT-14 - [RETROS] Implement one final suggestion to make our process better TO DO

Like Be the first to like this retrospective

The specified fields should be completed accordingly:

1. **Page name** - it is good practice to name the page of the Sprint Retrospective note with the following schema: *date(YYYY-MM-DD) + sprint name + "Retrospective"*
2. **Date** - select the date of the retrospective from the calendar
3. **Participants** - list all of the participants with @ prefix for an active link
4. **Content** - enter the outcome of the retrospective, such as boards, images, photos, lists, notations - anything your team used to conduct the retro
5. **Actions & Decisions** - create improvement tasks in Jira and add them here to a checklist or include a list of decisions that were made during or as a result of the retro

7.3 Versioning and release notes

A version is a set of features and fixes released together as a single update to your application. Assigning issues to versions helps to plan the order in which new features (stories) and bugfixes for your application will be released to your customers.

- (i)** In Jira Software, versions represent points-in-time for a project. They help you organize your work by giving you milestones to aim for. You can assign issues in your project to a specific version, and organize your sprints around completing work in that version.

7.3.1 Managing versions

1. Go to the **Backlog** of your Scrum project.
2. Click **VERSIONS** on the left side of the board (aligned vertically) to open it.

Add a new version	Click Create version (you will need to hover over the 'VERSIONS' panel to show this link), enter the version details, and create it. <ul style="list-style-type: none"> • The Start Date is used to give you a more accurate Version Report⁷³ in cases where you might plan a version many weeks or even months in advance, but not actually commence work until closer to the release date. • The End Date is used to calculate the days remaining in a release on the Release Hub⁷⁴.
Update a version's details	For the version name, click the arrow next to the name, then choose Edit name . For other fields (e.g. Description), click the field to edit it.
Add an issue to a version	Drag and drop the issue onto the version in the 'VERSIONS' panel.
Remove an issue from a version	Drag and drop the issue onto Issues without versions in the 'VERSIONS' panel.
Filter issues by version	Click the version in the 'VERSIONS' panel to show only issues in that version. Click All issues to remove the filter. <i>Alternatively, click Clear all filters next to the sprint's name.</i>

⁷³ <https://confluence.atlassian.com/jirasoftwarecloud/version-report-777001521.html>

⁷⁴ <https://confluence.atlassian.com/jirasoftwarecloud/using-the-release-page-to-check-the-progress-of-a-version-764478141.html>

- i** The blue horizontal bar under the version's name (in the 'VERSIONS' panel) indicates progress towards completing the work estimated for the version. Note that this bar only represents progress on the estimated issues in a version.

- i** You can create as many versions as you think is necessary. For example, you might create several versions, to plan ahead. Or you might just have one or two versions for now.

Once you create a version, the **Fix version** and **Affects version** fields will become available on your issues.

- i** **Affects version** is the version in which a bug or problem was found. Although it can be useful for tracking problems, it isn't used very often in Jira.
Fix version is the version where you plan on releasing a feature or bug-fix to customers. This field is used for release planning, monitoring progress and velocity, and is used widely in reporting. This is most likely the field you want.

The screenshot shows two Jira interfaces. On the left, a 'Backlog' view displays a list of issues: TRT-15 Test bug 2, TRT-16 Test bug 3, and TRT-17 Test bug 4. Each issue is associated with 'Version 1.1' and a 'Test epic'. On the right, a detailed view of 'Test_EG_Retail / TRT-15' shows the issue 'Test bug 2' with its status, priority, and assignee. A dropdown menu for 'Affects Version/s' is open, showing 'Version 1.1' and 'Unreleased Versions'.

7.3.2 Tracking and reporting releases

Once you have added all items to your release, there are two ways of generating reports: in Jira directly and in Confluence. You can run a release report in Jira at any time to see the progress of your release. This report will show you how many total issues are in the release broken down by the number in a completed status, in progress or yet to do. One nice advantage is the progress bar that shows this all visually.

The screenshot shows the Jira 'Releases' interface. On the left, a sidebar lists project navigation options like Main Board, Backlog, Active sprints, Releases (selected), Reports, Issues, and Components. The main area shows a summary of releases: Version 1.2 (15 total, 9 released, 12 unreleased) and Version 1.1 (15 total, 9 released, 12 unreleased). A progress bar indicates the status of each version. To the right, a detailed 'Releases' table lists the versions with their status, progress, start date, release date, and description. Buttons for 'Manage Versions', 'Merge versions', and 'Add' are available.

Version 1.1 UNRELEASED

Start: 01/Jul/20 Release: 31/Jul/20 [Release Notes](#)

The version number 1.1

1 day left

0 Warnings	5 Issues in version	2 Issues done	0 Issues in progress	3 Issues to do																																										
View in Issue Navigator																																														
<table border="1"> <thead> <tr> <th>P</th> <th>T</th> <th>Key</th> <th>Summary</th> <th>Assignee</th> <th>Status</th> <th>Development</th> </tr> </thead> <tbody> <tr> <td>=</td> <td>BUG</td> <td>TRT-8</td> <td>Test bug 1</td> <td>Unassigned</td> <td>DONE</td> <td></td> </tr> <tr> <td>=</td> <td>BUG</td> <td>TRT-15</td> <td>Test bug 2</td> <td>Unassigned</td> <td>TO DO</td> <td></td> </tr> <tr> <td>=</td> <td>BUG</td> <td>TRT-16</td> <td>Test bug 3</td> <td>Unassigned</td> <td>TO DO</td> <td></td> </tr> <tr> <td>=</td> <td>BUG</td> <td>TRT-17</td> <td>Test bug 4</td> <td>Unassigned</td> <td>TO DO</td> <td></td> </tr> <tr> <td>=</td> <td>BUG</td> <td>TRT-20</td> <td>Test bug 0</td> <td>Unassigned</td> <td>DONE</td> <td></td> </tr> </tbody> </table>					P	T	Key	Summary	Assignee	Status	Development	=	BUG	TRT-8	Test bug 1	Unassigned	DONE		=	BUG	TRT-15	Test bug 2	Unassigned	TO DO		=	BUG	TRT-16	Test bug 3	Unassigned	TO DO		=	BUG	TRT-17	Test bug 4	Unassigned	TO DO		=	BUG	TRT-20	Test bug 0	Unassigned	DONE	
P	T	Key	Summary	Assignee	Status	Development																																								
=	BUG	TRT-8	Test bug 1	Unassigned	DONE																																									
=	BUG	TRT-15	Test bug 2	Unassigned	TO DO																																									
=	BUG	TRT-16	Test bug 3	Unassigned	TO DO																																									
=	BUG	TRT-17	Test bug 4	Unassigned	TO DO																																									
=	BUG	TRT-20	Test bug 0	Unassigned	DONE																																									

7.3.3 Adding release notes

Release notes for selected version can be generated in TXT or HTML format.

[Configure Release Notes](#)

Bug

- [TRT-8] - Test bug 1
- [TRT-15] - Test bug 2
- [TRT-16] - Test bug 3
- [TRT-17] - Test bug 4
- [TRT-20] - Test bug 0

Edit/Copy Release Notes

The text area below allows the project release notes to be edited and copied to another document.

```
Release Notes - Test_EG_Retail - Version Version 1.1

<h2>      Bug
</h2>
<ul>
<li>[<a href='https://jira.eg.dk/browse/TRT-8'>TRT-8</a>] -      Test bug 1
</li>
<li>[<a href='https://jira.eg.dk/browse/TRT-15'>TRT-15</a>] -      Test bug 2
</li>
<li>[<a href='https://jira.eg.dk/browse/TRT-16'>TRT-16</a>] -      Test bug 3
</li>
<li>[<a href='https://jira.eg.dk/browse/TRT-17'>TRT-17</a>] -      Test bug 4
</li>
<li>[<a href='https://jira.eg.dk/browse/TRT-20'>TRT-20</a>] -      Test bug 0
</li>
</ul>
```

7.3.4 Release burndown chart

The release burndown chart shows how much work has been completed, and the total work remaining. Burndown charts are used to predict your team's likelihood of completing their work in the time available. They're also great for keeping the team aware of any scope creep that occurs. This report is available in **Reports** section of your project.



Note that to use the release burndown chart, you'll need to estimate your issues.



★ Release menu ★ Work added ★ Work remaining ★ Work completed ★ Project completion

7.3.5 Confluence reports

In Confluence, there are a couple of options: a **Status report** and a **Change log** that is best used for release notes. They are both done by creating a new page and then selecting the *Jira report* option.

Create Help

Select space Parent: dev

① 30 new items have been added X

JIRA report NEW
Communicate JIRA information in easy to read reports.

Next Close

Select report type

* — Change log

Create a report with a list of JIRA issues for a project, release or specific query.



Status report

Create a report with charts to communicate the status of your release with stakeholders.

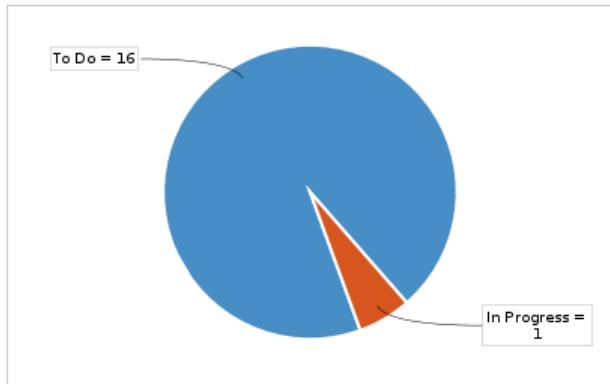
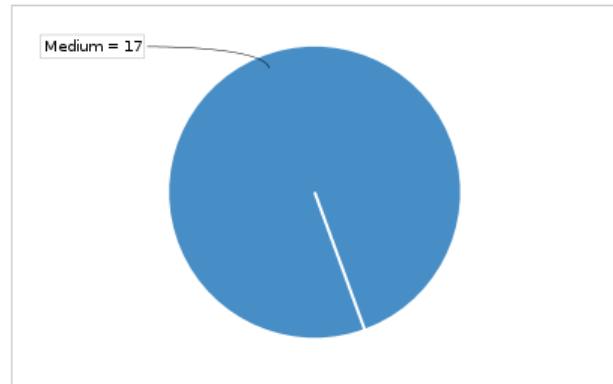
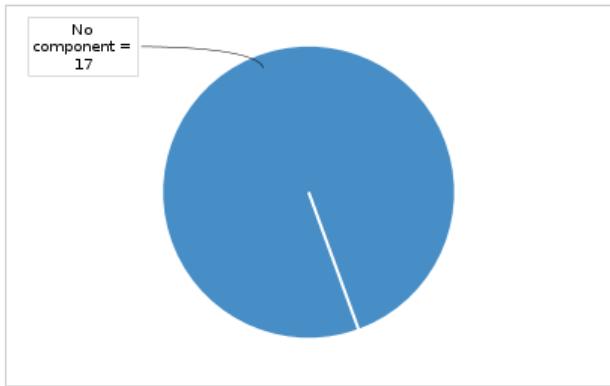
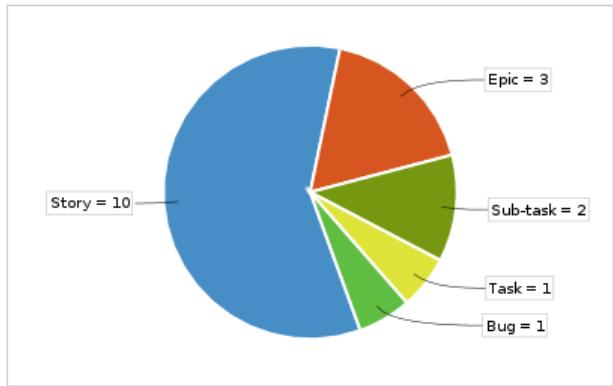
About JIRA reports

Communicate JIRA information in easy to read reports.

[Back](#)[Next](#)[Close](#)

7.3.5.1 Example of Status report:

Date	Nov 19, 2019
Issues	17 issues
Status	GREEN

Overall status for 17 issues**Priority****Component****Issue Type**

7.3.5.2 Example of Change log / release notes report:

Sample Scrum Dev Version 2.0

The screenshot shows a Jira Confluence page with the following sections:

- Page properties**: Date: Jan 16, 2018; Issues: 10 issues.
- Summary**: Insert release summary text here.
- Important highlights from this release**: 1. Highlight 1, 2. Highlight 2, 3. Highlight 3.
- All updates for this release**
- Epic**: SCRUMDEMO-27 [TO DO] Epic of epic proportions
- Bug**: SCRUMDEMO-13 [TO DO] As a developer, I can update details on an item using the Detail View >> Click the "SCRUMDEMO-13" link at the top of this card to open the detail view
SCRUMDEMO-8 [TO DO] As a product owner, I'd like to include bugs, tasks and other issue types in my backlog >> Bugs like this one will also appear in your backlog but they are not normally estimated
- Sub-task**: SCRUMDEMO-12 [TO DO] When the last task is done, the story can be automatically closed >> Drag this task to "Done" too
SCRUMDEMO-11 [IN PROGRE...] Update task status by dragging and dropping from column to column >> Try dragging this task to "Done"

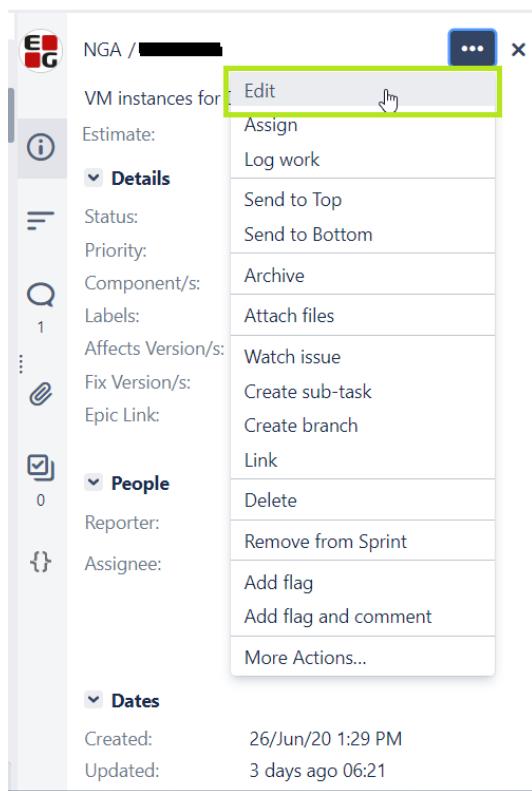
7.3.6 7.3.1 - Documenting Release Notes

For the purpose of smooth and optimal release notes creation, you can follow these steps presented below.

All issues that need to be included in the release notes document should have assigned fix version. It is not a mandatory field but you need to remember to fill it out for the purpose of release notes creation.

If the version number is not accessible your Jira Administrator can add it to your Project.

To change the fixVersion of the issue, first, you have to go to the edit mode. After that, the settings window will be prompted where the fixVersion space can be found and set. Example screenshots below.



Edit Issue : NG-1856

Configure Fields ▾

Assignee	[REDACTED]
Assign to me	
Attachment	<input type="button" value="Drop files to attach, or browse."/>
Reporter*	[REDACTED]
Start typing to get a list of possible matches.	
Linked Issues	blocks
Issue	<input type="button" value="+"/> <input type="button" value=""/>
Begin typing to search for issues to link. If you leave it blank, no link will be made.	
Fix Version/s	<input type="button" value=""/>
Start typing to get a list of possible matches or press down to select.	
Priority	Medium
Labels	<input type="button" value=""/>
Begin typing to find and create labels or press down to select a suggested label.	
Due Date	<input type="button" value=""/>
Story Points	<input type="button" value=""/>
Measurement of complexity and/or size of a requirement.	

FixVersion can be created anytime if the available fix versions are not fulfilling the needs.

Project settings

Releases

Version name: NGA 1.1 | Status: RELEASED | Progress: 100% | Start date: 20/Apr/20 | Release date: 20/Apr/20 | Description: Changes to Jira workflow and sprint naming convention | Actions: ...

After naming and saving a new version you should decide if an issue should be included in Client Release Notes or either in Technical Release Notes.

This operation should be done for every issue that will be included in a specific release.

PD-34

Big story with sub-tasks Register work in AX

Attachment: Attach | Create subtask | Link issue | Link page | ...

Description: Add a description...

Client Release Notes
content for client resale notes

Technical Release Notes
content for technical resale notes

Subtasks
0% Done

- PD-35 Task 1 development (TO DO)
- PD-36 Task 2 Validation (IN PROGRESS)

To Do

Assignee: Cezary Perendyk
Reporter: Cezary Perendyk
Development: Create branch
Labels: None
Activity: None

Include in client release notes: Yes
Include in technical release notes: Yes

Issue category: New features and functionality

IT does not affect release notes document

creation but you can release-ready version from Releases screen in Jira.

The screenshot shows the Jira interface for a project named "pipeline demo". The left sidebar contains links for PD board, Backlog, Active sprints, Reports, Releases (which is highlighted with a green border), Issues and filters, Pages, Components, Epics map, Add item, and Project settings. The main area displays a release for "Version 3" which is currently "UNRELEASED". It shows 7 issues in the version, with 1 issue done, 1 issue in progress, and 5 issues to do. Below this is a detailed list of 7 issues, each with its key, summary, and status. A "Release" button is located in the top right corner.

Release 3

Remaining issues

6 unresolved issues

Ignore and proceed with release

Move issues to version:

2

Release date



After the new version is set up and you have initial issues added to the newly created version you should navigate to your project Confluence workspace.

New Release Notes document is possible to be added by navigation to Jira reports section and clicking Add Jira Report button.

The screenshot shows a Jira workspace titled 'pipeline demo'. In the left sidebar, under 'Area', 'JIRA reports' is selected. The main content area displays a table of Jira issues with columns for 'Title', 'Date', 'Issues', and 'Status'. The 'Status' column has a green box around the 'Add Jira Report' button. The table data is as follows:

Title	Date	Issues	Status
pipeline demo version 2	Jan 08, 2020	0 issues	
pipeline demo 3	Jan 02, 2020	0 issues	
pipeline demo project, release notes for version 2	Jan 02, 2020	0 issues	
pipeline demo project, release notes for version 1	Jan 02, 2020	3 issues	

Afterward, you should choose Change log report and click Next button

Select report type

The screenshot shows a 'Select report type' dialog. It lists two options: 'Change log' and 'Status report'. The 'Change log' option is highlighted with a green box. The 'About JIRA reports' section provides a brief description of what a JIRA report is. A large 'Next' button is at the bottom right.

Change log
Create a report with a list of JIRA issues for a project, release or specific query.

Status report
Create a report with charts to communicate the status of your release with stakeholders.

About JIRA reports
Communicate JIRA information in easy to read reports.

In the next step, you should select the project, fix version and switch to the advanced

query editor

project = "<PROJECT IDENTIFIER>" AND fixVersion = "<VERSION IDENTIFIER>" AND "Include in client release notes" = Yes AND type != Sub-task

Create a change log

Project * pipeline demo

Fix version(s) Version 1

[Switch to advanced](#)

Title pipeline demo Version 1

About change logs

Keep a log of your teams progress or communicate deliverables. Generate a static or dynamic list of JIRA issues from a saved search, JIRA URL or JQL query.

Back [Create](#) Close

In Jira Query field you should enter a query with the pattern presented below where

project = "<PROJECT IDENTIFIER>" AND fixVersion = "<VERSION IDENTIFIER>" AND "Include in client release notes" = Yes AND type != Sub-task

PROJECT IDENTIFIER is your Jira project unique identifier

VERSION IDENTIFIER is your version unique identifier

For technical release notes Jira query should follow the pattern below

project = "<PROJECT IDENTIFIER>" AND fixVersion = "<VERSION IDENTIFIER>" AND "Include in technical release notes" = Yes AND type != Sub-task

Create a change log

JIRA Query * project = 'PD' AND fixVersion in ('Version 1')

[Switch to simple](#)

Title pipeline demo Version 1

Automatically update issue status and summaries from JIRA

About change logs

Keep a log of your teams progress or communicate deliverables. Generate a static or dynamic list of JIRA issues from a saved search, JIRA URL or JQL query.

Back [Create](#) Close

After selecting the query, the page template will occur, where the user should fill out

the Summary and Highlights. Please keep in mind that every new page/Release Note has EG logo attached in the right top corner. Page contact can be changed only in the edit mode. The logo can be changed, by replacing it with the proper one. The content of all the update table could be modified and adjusted to specific project needs.

Dashboard / ... / NGA Release notes

NGA 1.1 Test

Created by Łukasz Morecki, last modified 3 minutes ago.

Date Aug 03, 2020

Issues 3 issues

Summary
Summary a comprehensive and usually brief abstract, recapitulation, or compendium of previously stated facts or statements.

Important highlights from this release

1. Highlight of the issue 1
2. Highlight of the issue 2
3. Highlight of the issue 3

All updates for this release

T	Key	Summary	Assignee	Reporter	P	Status	Resolution	Created	Updated	Due
NG-1156	Jira workflow change: remove restriction on transition from InProgress to Ready status on all issue types	Dariusz Słęk	Dariusz Słęk	---	DONE	Done	Mar 11, 2020	Apr 21, 2020		
NG-1155	Jira workflow change: enable transition to Blocked status from all non-terminal statuses	Dariusz Słęk	Dariusz Słęk	---	DONE	Done	Mar 11, 2020	Apr 21, 2020		
NG-1091	Evaluate modifying the sprint naming convention to include year and reset sprint number each year	Dariusz Słęk	Dariusz Słęk	---	DONE	Done	Mar 04, 2020	Apr 02, 2020		

3 issues [jirareport](#)

Below pictures shows how and where the user can change the Jira Issue Filter.

Summary
Summary a comprehensive and usually brief abstract, recapitulation, or compendium of previously stated facts or statements.

Important highlights from this release

1. Highlight of the issue 1
2. Highlight of the issue 2
3. Highlight of the issue 3

Edit [View in Jira](#) [Remove](#)

Insert JIRA Issue/Filter

Search: project = 'PD' AND fixVersion in ('Version 1')

Search using any issue key, search URL, JIRA link, JQL, plain text or filter

<input checked="" type="checkbox"/> Key	Summary
<input checked="" type="checkbox"/> PD-41	Small stand-alone story
<input checked="" type="checkbox"/> PD-37	Medium story with sub-tasks Time regi*****--
<input checked="" type="checkbox"/> PD-35	Task 1 development
<input checked="" type="checkbox"/> PD-34	Big story with sub-tasks Register work in AX
<input checked="" type="checkbox"/> PD-33	some release notes test
<input checked="" type="checkbox"/> PD-32	new story

Display options

Select Macro Hint: type "Ctrl+Shift+J" in the editor to quickly access this dialog.

Insert **Cancel**

Once the Title, Summary, and Highlights are filled out, Logo corresponds to your business unit, Jira filter/table has a proper query, then the release notes document is ready to be released.

7.4 7.4 - Dashboard metrics description

7.4.1 Issue statistics widget

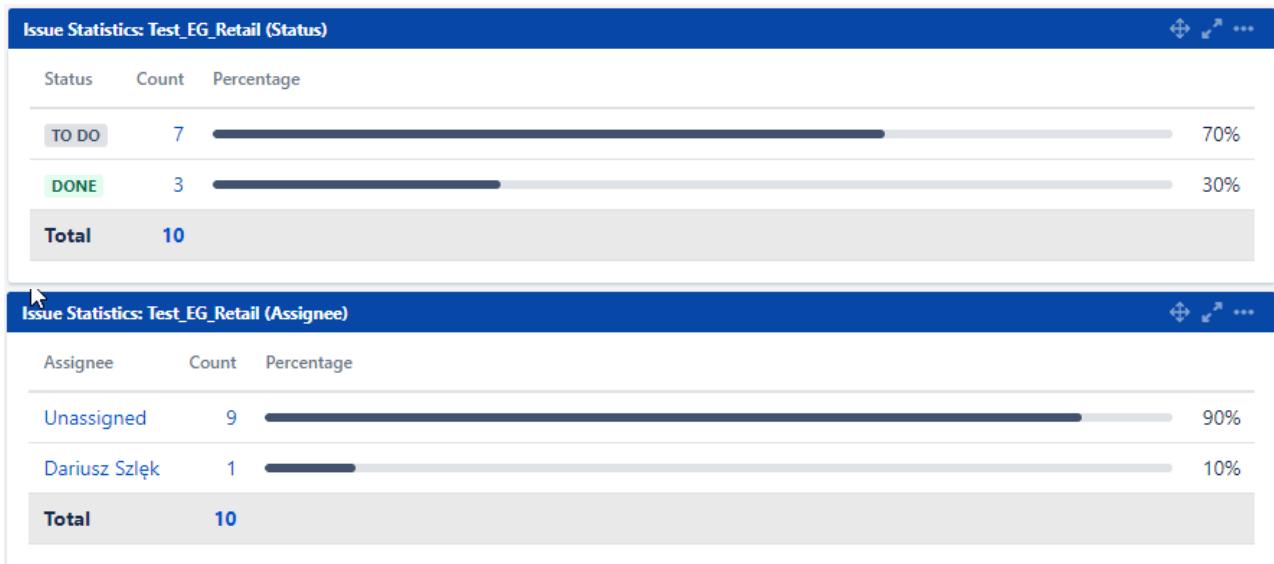
The screenshot shows the 'Issue Statistics' configuration page in Jira. At the top, it says 'Project or Saved* No Filter/Project selected'. Below that is a 'Filter:' search bar with placeholder text 'Search'. A note below the search bar says 'Project or saved filter to use as the basis for the graph.' and a link to 'Advanced Search'. Under 'Statistic Type:', the dropdown is set to 'Assignee'. A note below it says 'Select which type of statistic to display for this filter.'. Under 'Sort By:', the dropdown is set to 'Natural'. A note below it says 'Sort by row total or natural field order.'. Under 'Sort Direction:', the dropdown is set to 'Ascending'. Under 'Show Resolved Issue Statistics', the dropdown is set to 'No'. A note below it says 'Include resolved issues in the set of issues from which statistics are calculated.'. Under 'Auto refresh', there is a checkbox labeled 'Update every 15 minutes'. At the bottom right is a 'Save' button.

The widget allows for configuration of a 1-dimensional statistic bar chart based on a Jira issue filter and a selected statistic type:

- *Project or Saved filter* - defined Jira issue filter or Jira project for specifying the set of issues in scope of the statistics
- *Statistic type* - Statistic which is desired to be measured based on the set of issues from the indicated filter or project



Examples from Jira corporate metrics dashboard:



The widgets show (A) all issues with respect to the issue status (*Statistic type*) and (B) all issues in the project (*filters*) with respect to the issue assignee (*Statistic type*). The specific number count, percentage and total number of issues can be observed.

7.4.2 Pie chart widget

Pie Chart

Project or Saved* **No Filter/Project selected**

Filter:

Project or saved filter to use as the basis for the graph.
[Advanced Search](#)

Statistic Type*: **Assignee**

Select which type of statistic to display for this filter.

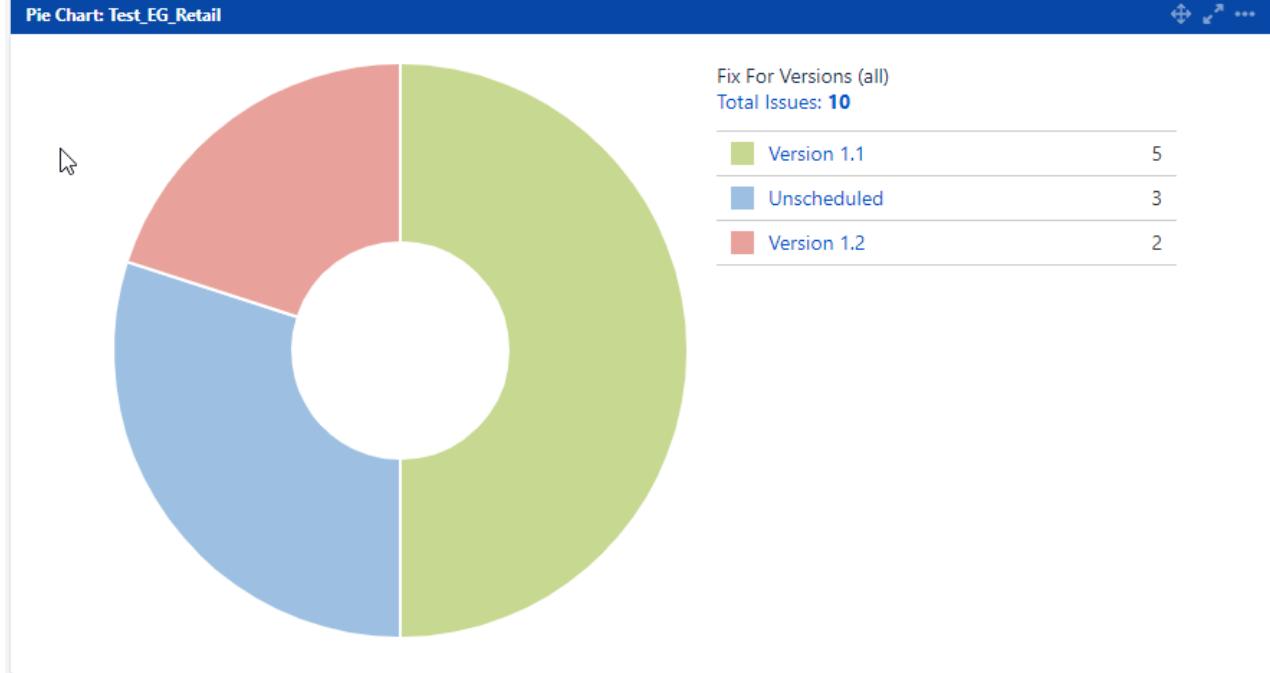
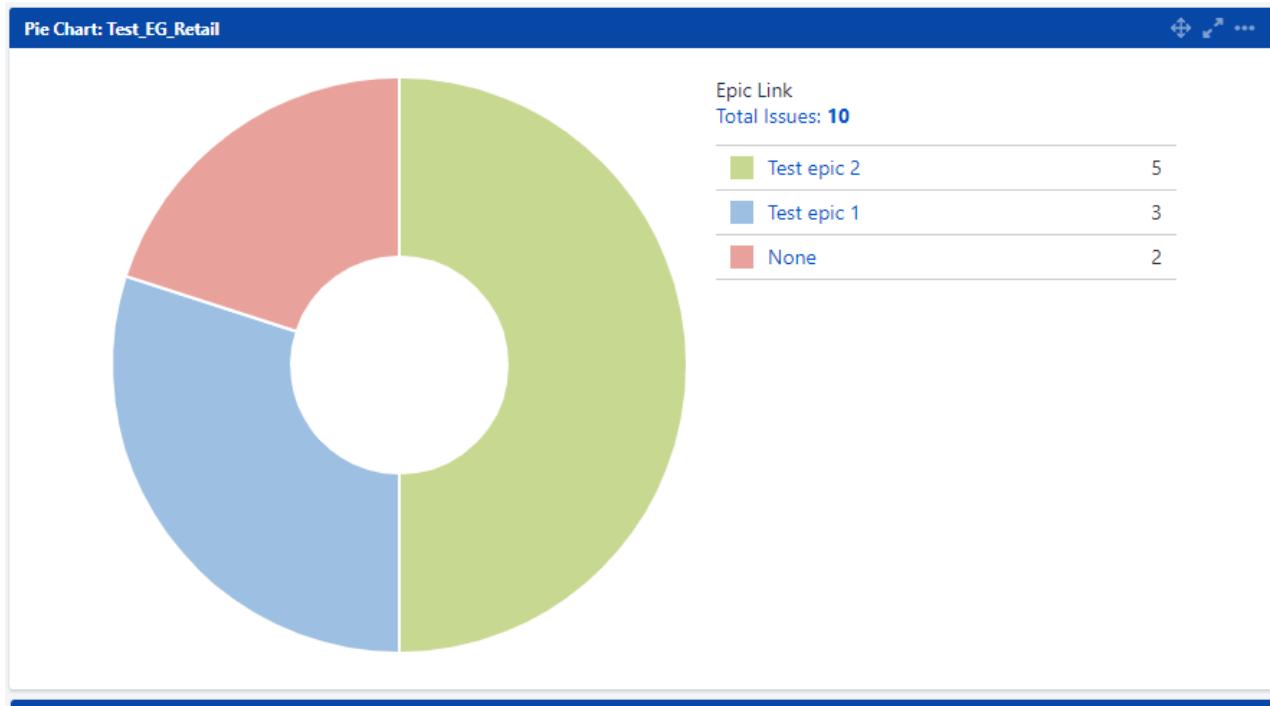
Auto refresh Update every 15 minutes

Save

The widget allows for configuration of a 1-dimensional statistic pie chart based on a Jira issue filter and a selected statistic type:

- *Project or Saved filter* - defined Jira issue filter or Jira project for specifying the set of issues in scope of the statistics
- *Statistic type* - Statistic which is desired to be measured based on the set of issues from the indicated filter or project

(i) Examples from Jira corporate metrics dashboard:



The widgets show (A) all issues with respect to the Epic assignment (*Statistic type*) and (B) all issues in the project (*filters*) with respect to the fixVersion (*Statistic type*). The specific number count with distinct coloring and total number of issues can be observed.

7.4.3 Two dimensional filter statistics widget

The screenshot shows the configuration interface for a 'Two Dimensional Filter Statistics' widget. The interface includes:

- Saved Filter:** A dropdown menu.
- XAxis:** Set to 'Assignee'.
- YAxis:** Set to 'Assignee'.
- Sort By:** Set to 'Natural'.
- Sort Direction:** Set to 'Ascending'.
- Number of Results:** A text input field containing '5'.
- Auto refresh:** A checkbox labeled 'Update every 15 minutes'.
- Save:** A button at the bottom left.

The widget allows for configuration of a 2-dimensional statistic chart based on a Jira issue filter and 2 selected statistic types:

- **Saved filter** - defined Jira issue filter for specifying the set of issues in scope of the statistics
- **XAxis** - First statistic which is desired to be measured based on the set of issues from the indicated filter or project; presented on the X-axis of the chart
- **YAxis** - Second statistic which is desired to be measured based on the set of issues from the indicated filter or project; presented on the Y-axis of the chart



Example from Jira corporate metrics dashboard:

The screenshot shows a Jira dashboard titled 'Two Dimensional Filter Statistics: Filter for EG Retail Main Board'. The dashboard displays the following data:

Status	Dariusz Szlek	Unassigned	T:
TO DO	0	7	7
DONE	1	2	3
Total Unique Issues:	1	9	10

Grouped by: Assignee

Showing 2 of 2 statistics.

The widgets show all issues in the project (*filter*) with respect to the status (*YAxis*) and assignee (*XAxis*). The specific number count for the cross-section of both statistics and total number of issues in either statistic can be observed.

7.4.4 Created vs. resolved chart widget

Created vs. Resolved Chart

Project or Saved* **No Filter/Project selected**

Filter: Project or saved filter to use as the basis for the graph.
Advanced Search

Period*: **Daily** The length of periods represented on the graph.

Days Previously* **30** Days in the past to collect data for the selected period.

Collection* **Count** Operation Progressively add totals (1.. 2.. 3), or show individual values (1.. 1.. 1).

Display the Trend of* **No** Unresolved Show the number of unresolved issues over time in a subplot.

Display Versions* **Only major versions** Show when versions were released on the chart.

Auto refresh Update every 15 minutes

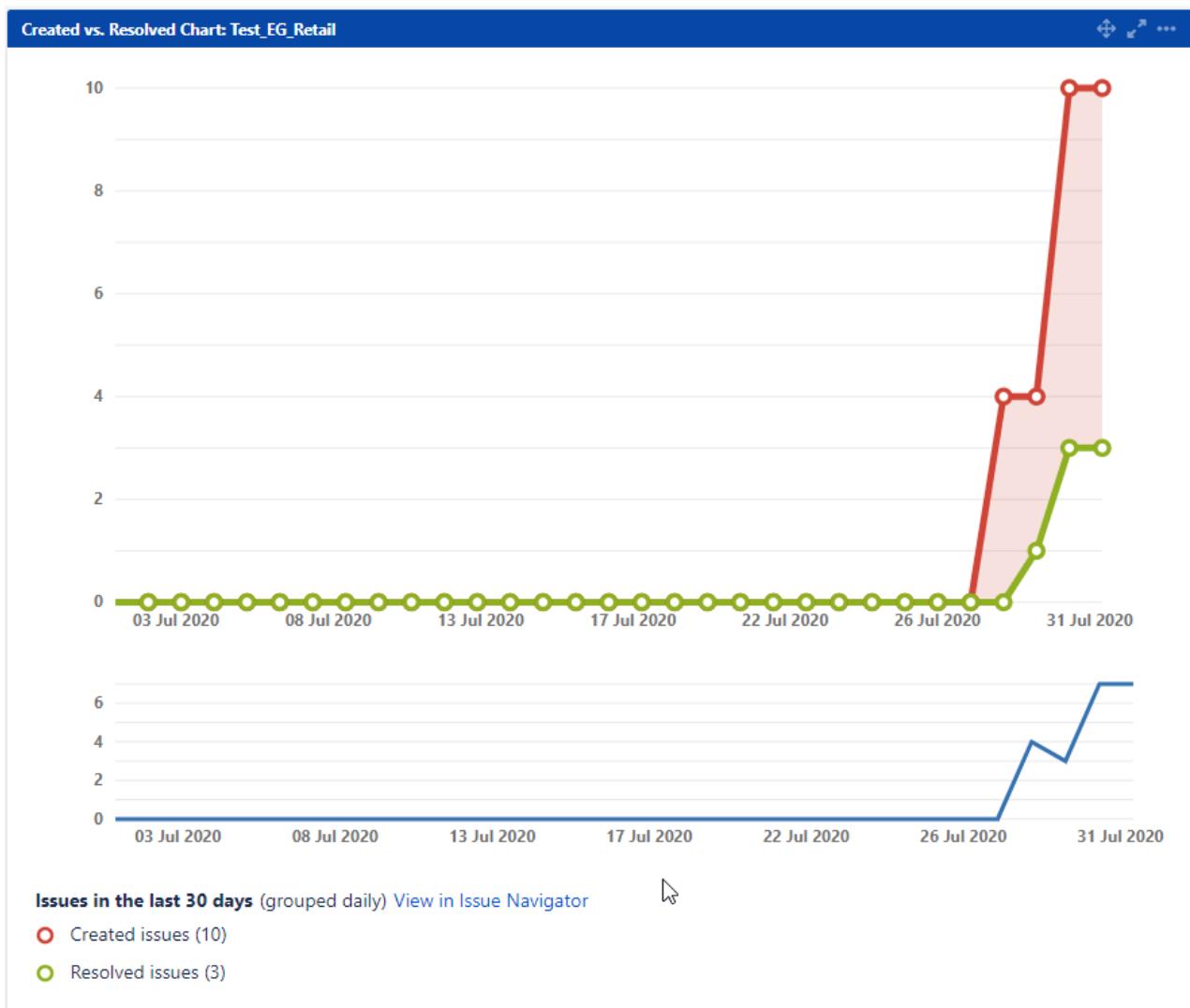
Save

The widget allows for configuration of a created/resolved defect graph based on a Jira issue filter and the defined time period:

- *Project or Saved filter* - defined Jira issue filter or Jira project for specifying the set of issues in scope of the graph
- *Period* - length of the periods (frequency of measurement)
- *Days previously* - days in the past taken into account for the measurement
- *Collection Operation* - incremental or individual
- *Display the Trend of Unresolved* - showing unresolved trend graph in addition
- *Display Versions* - showing released versions on the timeline



Example from Jira corporate metrics dashboard:



The widgets show all issues in the project (*filter*) with respect to the resolution and dates. The amount, dates of resolved/unresolved issues is shown; the trend line is also configured to be shown below with a grouping summary at the bottom.

7.5 7.5 - Logging time and transferring it to NetSuite ERP

After registering your worklogs in Jira you have a possibility to transfer them to the NetSuite ERP worksheet.

7.5.1 Logging time in Jira

To log time on story, task, bug or subtask, use the built-in Jira time logging features, by clicking log time in story menu:

The screenshot shows the Jira software interface. At the top, there is a navigation bar with links for Dashboards, Projects, Issues, Boards, and ERP, along with a 'Create' button. Below the navigation bar, a sidebar on the left displays a project named 'test' with three items: 'TEST board', 'Backlog', and 'Active sprints'. The main content area shows an issue titled 'my first issue' under the 'test / TEST-1' project. The issue has a blue icon and is labeled 'Story' with a priority of 'Medium'. A 'More' dropdown menu is open, showing options like 'Edit', 'Comment', 'Assign', 'Log work', and 'Agile Board'. The 'Log work' option is highlighted with a mouse cursor. On the right side, there are status and resolution fields.

In the log work dialog, provide the time spent and a description:

Log Work: TEST-1

Time Spent* (eg. 3w 4d 12h) [?](#)

An estimate of how much time you have spent working.

Date Started* [Calendar icon](#)

Remaining Estimate **Adjust automatically**
the estimate will be reduced by the amount of work done, but never below 0.

Use existing estimate of 0h

Set to (eg. 3w 4d 12h)

Reduce by (eg. 3w 4d 12h)

Work Description

Style [B](#) [I](#) [U](#) [A](#) [°](#) [♂](#) [♀](#) [≡](#) [☰](#) [⊕](#) [+](#)

writing a user manual

[Visual](#) [Text](#)

[Viewable by All Users](#)

[Find more time-tracking apps...](#)

[Log](#) [Cancel](#)

The worklogs on a specific Jira story/task/bug/subtask can be viewed on an issue view, when you switch to worklogs tab:

Activity

All Comments **Work Log** History Activity

 Jaroslaw Krochmalski logged work - 15/Jul/20 2:28 PM

Time Spent: 2h
writing a user manual

7.5.1.1 ERP Activity field

Each Jira issue (epic, story, bug, task, subtask) has a custom **ERP Activity** field which is being used for specifying the NetSuite ERP Project Task ID:

ERP Activity	ProjectTask-00000001234567
ERP Activity/Task ID	

If the ERP Activity is not present on the story or sub-task, Jira will search for the ERP Activity value on the parent story, parent Epic or parent story's Epic.

If it's not there and you don't know Project number ask your manager.

7.5.1.2 Additional Netsuite information

When you enter the ERP Activity value, Jira will query NetSuite to get additional task details, like NetSuite Project, Task and Customer:

Field Tab	GDPR	Release Notes
Team name:	A4 Timber	
Sprint:	A4 Timber - S70_23 (10/04), A4 Timber - S71_23 (24/04), A4 Timber - S72_23 (08/05), A4 Timber - S73_23 (22/05), A4 Timber - S74_23 (05/06), A4 Timber - S75_23 (19/06), A4 Timber - S76_23 (03/07), A4 Timber - S77_23 (17/07), A4 Timber - S78_23 (31/07)	
Issue category:	Customer funded development	
ERP Activity:	ProjectTask-00000002416649	
Netsuite Project:	Byggmakker Handel AS : Implementering Byggmakker 2023 (1267207)	
Netsuite Task:	Change 8 - Additional Task; General discount (Allmenning rabat), Development part	
Netsuite Customer:	Byggmakker Handel AS	

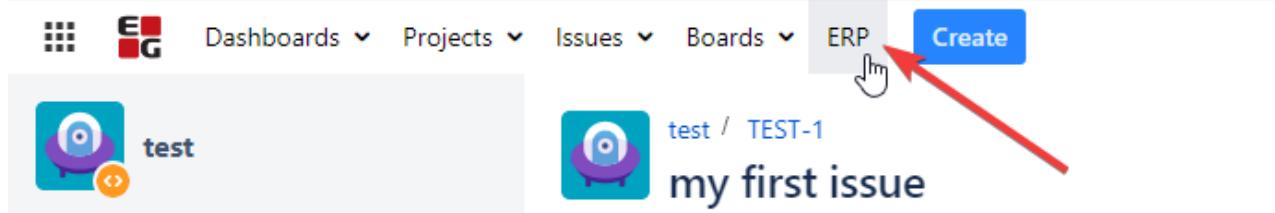
If there is no such activity in NetSuite, an error message will show up:

ERP Activity:	123
Netsuite Project:	Error: The project task is unknown

Note, that this is not preventing you to enter an incorrect project task ID into Jira. It's for informational purposes only.

7.5.2 Transferring logged time to NetSuite ERP

When you want your time to be transferred to NetSuite, click the **ERP** button in the main Jira menu:



The **My worklogs** main screen will open:

The screenshot shows the 'My worklogs' screen in Jira. On the left, there is a sidebar with tabs for 'My worklogs' (which is active) and 'Activities'. The main area displays a table of worklogs. The columns include: a checkbox, 'ERP' (which is checked), 'Issue' (with icons for Jira and NetSuite), 'Date started', 'Time spent (hrs)', 'Remaining', 'Activity' (with a dropdown arrow), 'Transaction Text', and 'Internal Text'. Two worklogs are listed:

	ERP	Issue	Date started	Time spent (hrs)	Remaining	Activity	Transaction Text	Internal Text
<input type="checkbox"/>			JAP-3	10/27/2022 13:30:00	1.5	18.5	ProjectTask-00000001234567	JAR Story 1
<input type="checkbox"/>			JAP-16	10/27/2022 13:31:00	2.25	7.75	ProjectTask-00000009999999	JAR Bug 2

At the bottom of the table, there are several buttons: 'Refresh', 'Send selected to ERP' (with a red circle labeled '2'), 'Send all to ERP' (with a red circle labeled '3'), 'Remove selected', 'Remove all', and 'Download CSV' (with a red circle labeled '5').

The ERP screen **My Worklogs** tab (1) will list all worklogs you have not yet transferred to NetSuite. Before sending them, your worklogs can be edited - fields **Activity**, **Transaction Text** and **Internal Text** are editable when you click them.

The **ERP Activity** value will be taken from the story/sub-task that the worklog is registered on, or if not present - from its parent, Epic, or parent's Epic. You can also edit the ERP Activity manually or choose from predefined list of your custom activities. You need to start typing the activity number or its description, for autocomplete dropdown to show up:

Activity	Transaction Text
000	JAR Story 1
ProjectTask-00000001234567	
ProjectTask-00000009999999	
ProjectTask-00000002424242	

When ready, you can either transfer selected (2) or all (3) worklogs to the NetSuite ERP worksheet. If you don't want to send a specific worklog to the NetSuite ERP worksheet, you can remove it from the list (4).

i Note that deleting a row on this screen will not delete the worklog on a Jira issue.

When transferred(sent) to the ERP, the entry will be removed from the list of worklogs.

Export your time to Netsuite per 5-10 logs to do not overload it.

! Note that Jira will put your worklogs into the NetSuite ERP worsheet. You still need to log into NetSuite ERP and submit your registered time in order to finalize it.

7.5.2.1 Custom activities

The custom personal activities available to choose on the autocomplete dropdown can be edited in the Activities tab:

The screenshot shows the Jira interface with the 'My worklogs' page selected. At the top, there is a navigation bar with links for Dashboards, Projects, Issues, Boards, ERP, and a blue 'Create' button. Below the navigation bar, the title 'My worklogs' is displayed. On the left, there is a sidebar with tabs for 'My worklogs' (selected) and 'Activities'. A red arrow points to the 'Activities' tab. The main content area shows a table with two rows. The first row has a checkbox, the word 'Issue', and a date/time column. The second row has a checkbox, the icon for 'TEST-1', and the text '44 minut'. A yellow callout box contains a warning message: '⚠ Note, that the list is **personal**. You will only see your own activities here and will be able to define activities for **yourself**'.

⚠ Note, that the list is **personal. You will only see your own activities here and will be able to define activities for **yourself**.**

It's a CRUD screen which allows adding, deleting and editing custom activities you can then use on the My Worklogs autocomplete dropdown:

The screenshot shows the 'My activities' screen. At the top, there is a header 'My activities' with a 'My worklogs' tab (selected) and an 'Activities' tab. The main content area is a table with columns for 'ID', 'Activity', and 'Description'. There are three rows of data: ID 2, Activity 'ProjectTask-00000001234567', Description 'Test activity 1'; ID 3, Activity 'ProjectTask-00000009999999', Description 'Test activity 3'; and ID 4, Activity 'ProjectTask-00000002424242', Description 'Test activity 2'. To the right of the table, there are 'Add', 'Delete', and 'Delete' buttons.

7.5.3 Logging working time using Worklogs Plugin

On main Jira tab choose tab Worklogs

The screenshot shows the Jira interface with the 'Worklogs' tab selected. At the top, there is a navigation bar with links for Dashboards, Projects, Issues, Boards, Plans, ERP, Tests, and 'Worklogs' (selected). A red arrow points to the 'Worklogs' tab. The main content area is currently empty.

On Worklog screen choose



The screenshot shows the Jira Worklogs interface. At the top, there are filters for 'Project or Filter' (All projects), 'User or Group' (Sergii Koval), and a date range from '11/01/2021' to '11/30/2021'. Below the filters, there are grouping options: 'Period grouping' (Day), 'Categorize by' (User), 'Group by' (Project), and 'Secondary Grouping' (None). A 'View Report' button is also present. The main area displays a table for Sergii Koval, showing a single entry for November 1st with a total of 0 hours worked. The 'Log time' button, located in the top right corner of the table header, is highlighted with a large red arrow pointing directly at it. The table has columns for User, Project, Total, and dates from Mon 01 Nov to Fri 12 Nov. At the bottom right of the table, there is a 'Show 10 entries' dropdown.

You will see a new pop-up window. Fill it, provide issue key and worked time, click on Log time button

⌚ Log time

User



Sergii Koval



Issue

NG-3851



NG-3851

Worklogs - Time tracking and reports extension for...

Worked

2w 4.5d 6h 45m

Remaining estimate

Original estimate

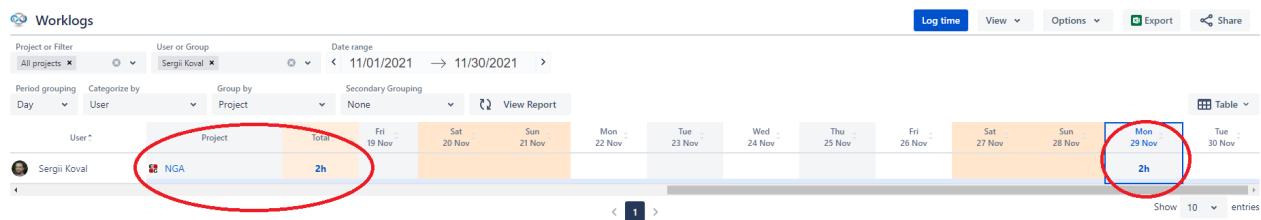
Work description

Aa **B** I ... A “ —

Log time

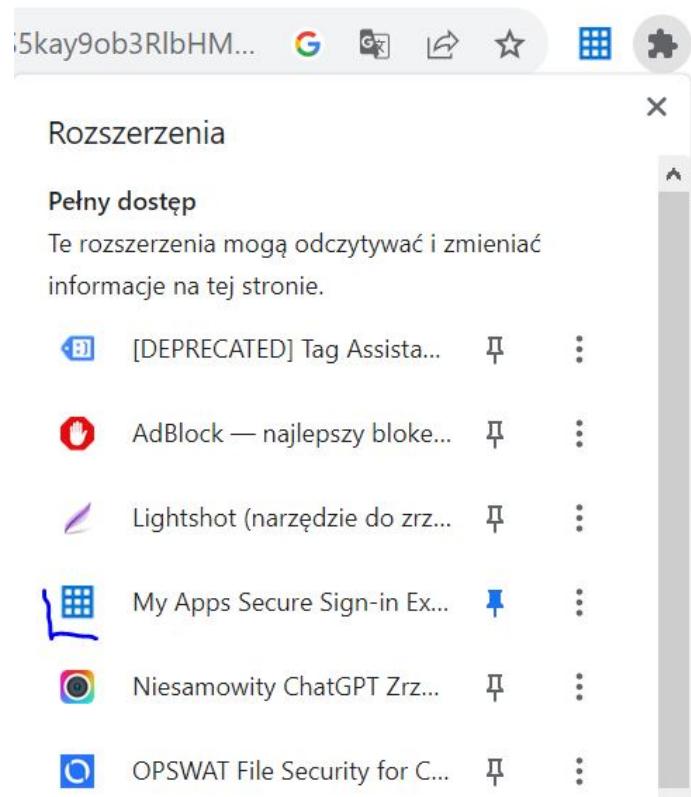
Cancel

You will see the next result

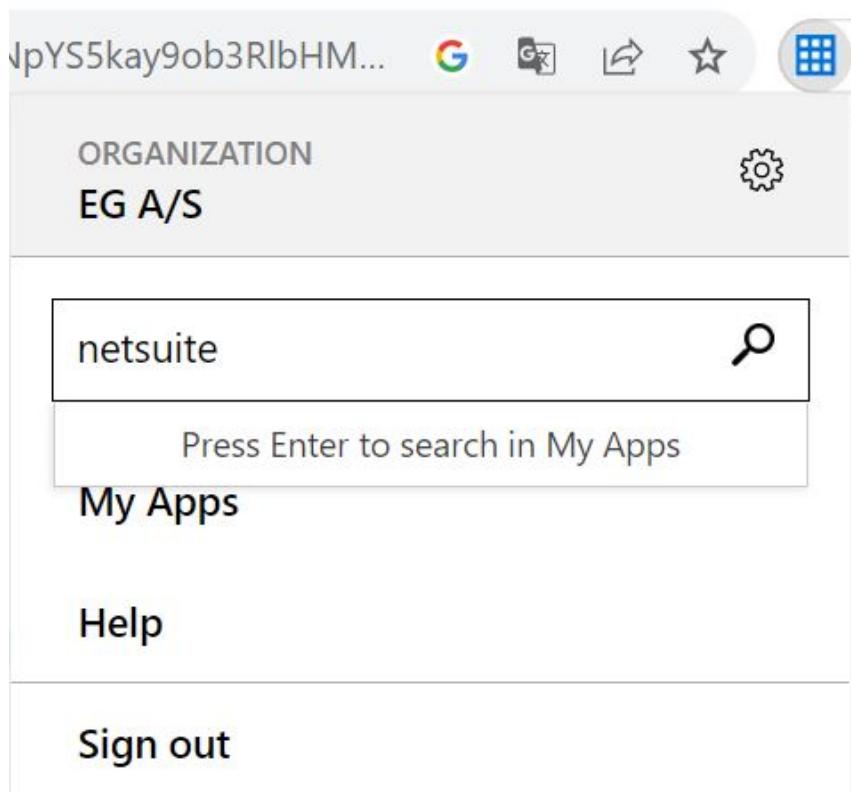


7.5.4 Submitting work hours using Netsuite Chrome plugin

Open in Chrome browser "My Apps" Extensions



Open in "My Apps" extension "Netsuite" app



Log in with EG account

The screenshot shows the Microsoft My Apps dashboard. At the top, there is a search bar with the text 'netsuite' and a placeholder 'Showing results for 'netsuite''. Below the search bar, there are buttons for 'Add apps', 'Create collection', and 'Customize view'. The main area displays a grid of app tiles. The visible tiles include:

- Add-Ins
- Bookings
- Calendar
- Clipchamp
- Delve
- EG - NetSuite OpenAir - PROD
- EG - Paybooks SSO
- EG Netsuite
- EG TrueLink
- EG Zalaris
- Engage
- Excel

Click "Weekly timesheet"

Check if imported hours from Jira are correct and submit each week separately

7.6 7.6 - User management

This guide will prepare you to being able to manage users of Jira and Confluence.

7.6.1 Attention!

Before being able to conduct the following steps you must have the Jira Project Administrator role. To get this please contact the NGA team.

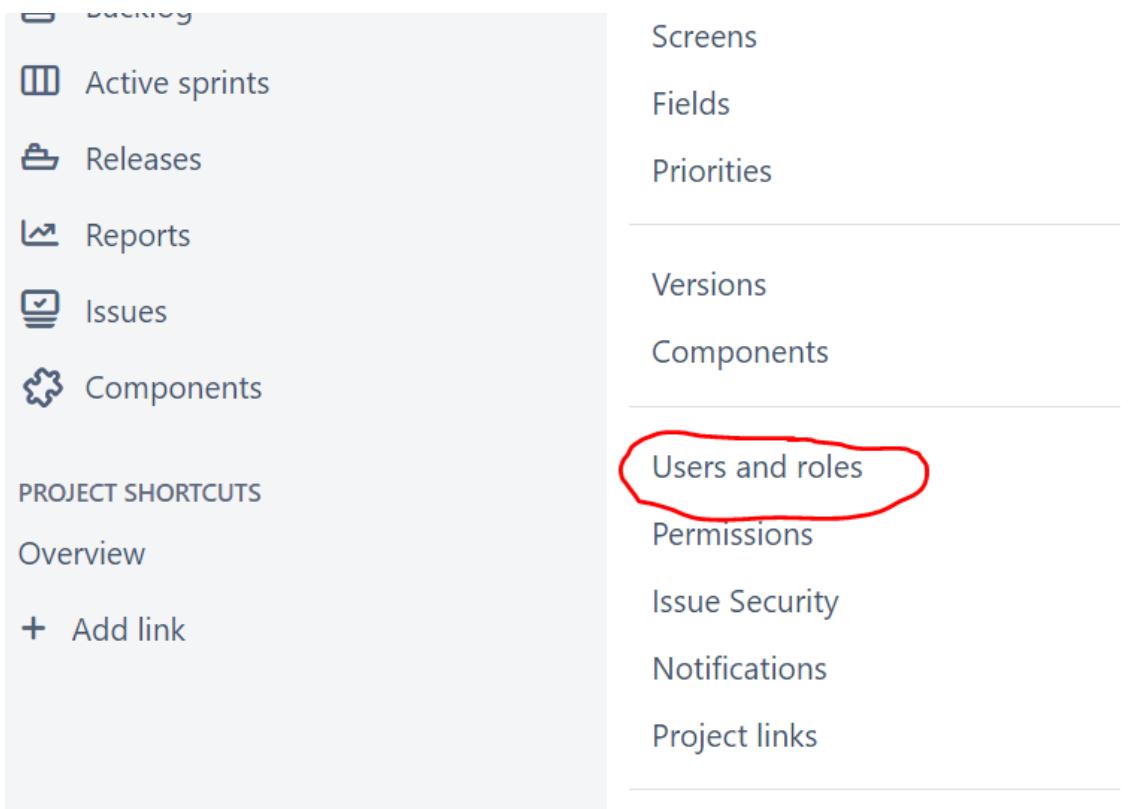
General preparation before doing any changes. Navigate to the right project using the project picker at the top of the screen.



Click Project Settings in the bottom left corner

A screenshot of the Jira Project Settings sidebar. On the left, there's a sidebar with icons and links: 'Issues' (selected), 'Components', 'PROJECT SHORTCUTS' (with 'Overview' and '+ Add link' options), and 'Project settings' (which is circled in red). On the right, under 'Issue types', there's a list of issue types: Bug, Business Project, Epic, Story, Sub-task, and Task. At the bottom of the sidebar, there are '... more' and 'View all' buttons.

And select Users and Roles



Note: If you are not able to see the Project Settings button you most likely need to be added the Project Admin role.

7.6.2 Creating a new user

When you have navigated to the user management in Jira you can click on the Add users to a role-button to add new users.

Users and roles

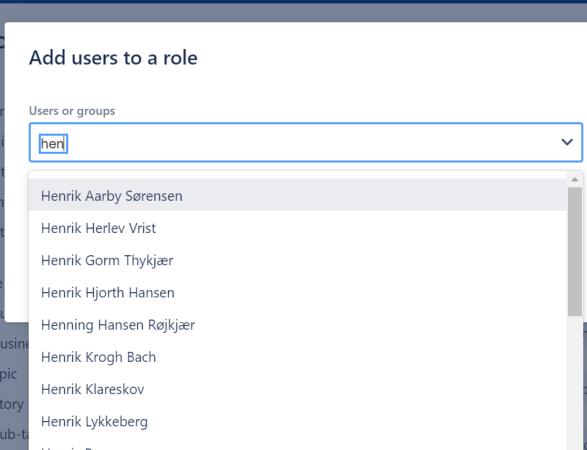
Project lead  Linda Ammitzbøll Default Assignee Unassigned

Add users to a role

Edit defaults

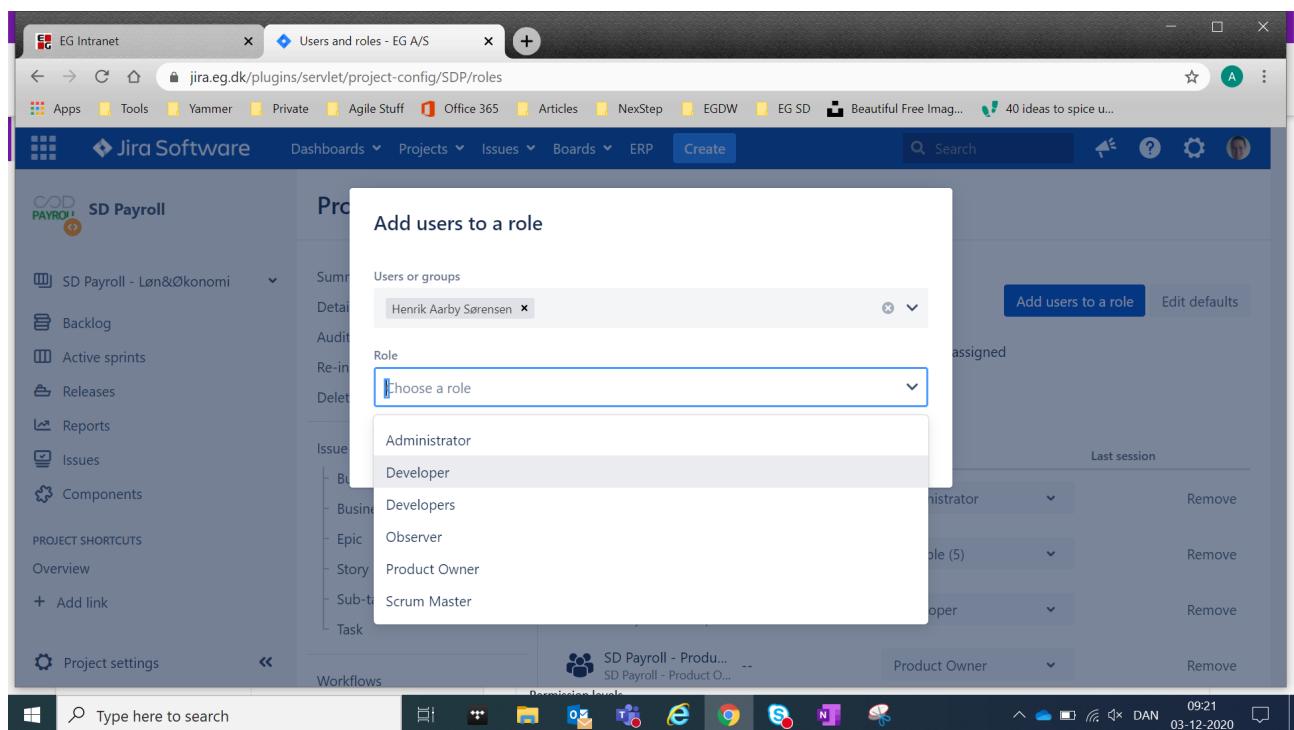
Name	Email address	Roles	Last session
 SD Payroll - Projec... SD Payroll - Project A...	--	Administrator	Remove
 jira-administrators jira-administrators	--	Multiple (5)	Remove
 SD Payroll - Devel...	--	Developer	Remove

In the dialogue that opens start typing the name of the user that needs access to Jira. When you start typing Jira will suggest a list of users that are part of the AD group that can get access to Jira.



The screenshot shows the 'Add users to a role' dialog box open over a Jira interface. The search input field contains the text 'herl'. Below it, a list of suggested users is displayed, including Henrik Aarby Sørensen, Henrik Herlev Vrist, Henrik Gorm Thylkær, Henrik Hjorth Hansen, Henning Hansen Røjkjær, Henrik Krogh Bach, Henrik Klareskov, Henrik Lykkeberg, and Henrik Berg. The background shows the Jira navigation bar and a project page for 'SD Payroll'.

Note: If the user that you would like to add doesn't appear in the list, then please contact the NGA team and request them to add the user to the AD group who has got access to the Atlassian tools in EG.



Select the right role for the user, and a click the Add-button. Now the user will have access to the project you are in with the role you have chosen.

7.6.3 Changing the role of a user

If you need to update a user with a new role, then search out the user in the search field found a the top of the user management screen in Jira.

The screenshot shows the Jira Software interface for the 'SD Payroll' project. On the left, there's a sidebar with project navigation options like 'Backlog', 'Active sprints', 'Releases', 'Reports', 'Issues', 'Components', and 'Project settings'. The main area is titled 'Project settings' and contains sections for 'Summary', 'Details', 'Audit log', 'Re-index project', and 'Delete project'. Below these is a 'Users and roles' section. A search bar at the top of this section has 'henrik' typed into it. To the right of the search bar are buttons for 'Add users to a role' and 'Edit defaults'. The table below lists three users: Henrik Krogh Bach (Administrator), Henrik Nørgård (Developer), and Henrik Skovsgaard (Developer). Each user row includes their name, email address, current role, last session date and time, and a 'Remove' button.

Name	Email address	Roles	Last session	Action
Henrik Krogh Bach hekba@eg.dk	hekba@sd.dk	Administrator	17/11/20 09:58 AM	Remove
Henrik Nørgård hennno@eg.dk	hennno@sd.dk	Developer	3 days ago 08:43 AM	Remove
Henrik Skovsgaard hespe@eg.dk	hespe@sd.dk	Developer	19/11/20 13:00 PM	Remove

Next to the person you can select the role that the user should be updated to.

7.6.4 Removing access for a user

Name	Email address	Roles	Last session	Action
Henrik Krogh Bach hekba@eg.dk	hekba@sd.dk	Administrator	17/11/20 09:58 AM	Remove
Henrik Nørgård hennno@eg.dk	hennno@sd.dk	Developer	3 days ago 08:43 AM	Remove
Henrik Skovsgaard hespe@eg.dk	hespe@sd.dk	Developer	19/11/20 13:00 PM	Remove

In the search field type the name of the user that needs to be remove from the project. Then click on Remove in the last column of the row. This will remove the user from the current project you are in. Repeat the steps for all projects if needed.

7.6.5 Permission levels

An overview of the permissions the different roles have please see this [Permission Scheme \(see page 373\)](#).

This guide will prepare you to being able to manage users of Confluence.

7.6.6 Attention!

Before being able to conduct the following steps you must have the Jira Project Administrator role. To get this please contact the NGA team.

General preparation before doing any changes. Navigate to the right Confluence Space using the Space picker at the top of the screen.

SD Payroll

Pages

Blog

SPACE SHORTCUTS

JIRA: SD Payroll

Retrospectives

PAGE TREE

Welcome to your software project space

This is home base for your software project. It's where you can store important documentation, and discuss it.

Next, you might want to:

- Customize the home page - Click **Edit**
- Create more pages - Hit **Create** in the header

In the left-hand menu click Space Tools at the bottom of the screen and select the Permissions.

Overview

Permissions

Content Tools

Look and Feel

Audit log

Integrations

Reorder pages

Configure sidebar

Space tools

Note: If you are unable to navigate to the Permissions screen you might not have sufficient permissions to do so. Contact your Space Admin or NGA to get the needed permissions setup.

In the Permissions screen you can manage permissions for employees who already have access, add employees that need access and remove employees that shouldn't have access to a specific Space anymore.

Please note that if an employee is part of multiple Spaces their permissions will only be updated for the Space that you are currently working with. If all permissions should be updated, you'll need to navigate to all the Spaces and do the updates of permission for each Space.

To manage permissions for employees please navigate to the section for Individual Permissions. At the end of this section click on the button, Edit Permissions.

The screenshot shows a table of user permissions. The columns represent different permissions: View, Delete Own, Add, Delete, Add, Delete, Add, Delete, Add/Delete, Delete, Export, and Admin. The rows list individual users: Marcin Domanski (xxycf@eg.dk). The 'Edit Permissions' button is highlighted with a red oval.

Marcin Domanski (xxycf@eg.dk)		View	Delete Own	Add	Delete	Add	Delete	Add	Delete	Add/Delete	Delete	Export	Admin
		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				

This will enable the configuration of permission for the individual users. You simply tick the boxes with the permissions that the employee(s) should have.

Individual Users

Grant permissions to individual users, regardless of which groups they are a member of.

	All		Pages		Blog		Attachments		Comments		Restrictions		Mail	Space
	View	Delete Own	Add	Delete	Add	Delete	Add	Delete	Add	Delete	Add/Delete	Delete	Export	Admin
Alexandra Theiss Janum-Szatkowski (altja@eg.dk)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Amos Sejer Hoste (amsho@eg.dk)	<input checked="" type="checkbox"/>													
Andreas Joensen (anjoe@eg.dk)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Note, to see what type of access is usually granted for different roles in EG please visit the [Permission Scheme⁷⁵](#) overview here.

If the employee is not listed in the overview, then navigate to end of the section again and use the search field to find the employee.

⁷⁵ <https://confluence.eg.dk/display/NG/Permission+Scheme>

Marcin Domanski (xxycf@eg.dk)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
Select All											
										<input type="button" value="Search"/>	<input type="button" value="Add"/>

Note, if you cannot find the employee via the search field, you need to contact the NGA Team so they can add the employee to the right Active Directory group that has got access to Jira and Confluence.

7.7 7.7 - Instruction to GDPR and threat assessment

7.7.1 Purpose

The instruction describes how to identify and assess GDPR requirements, threats and risks related to software solution.

7.7.2 Scope

The instruction is used in the Risk Assessment Process, Development Process, Software Development Process and the Software Change Process.

7.7.3 Instruction

7.7.3.1 Change type:

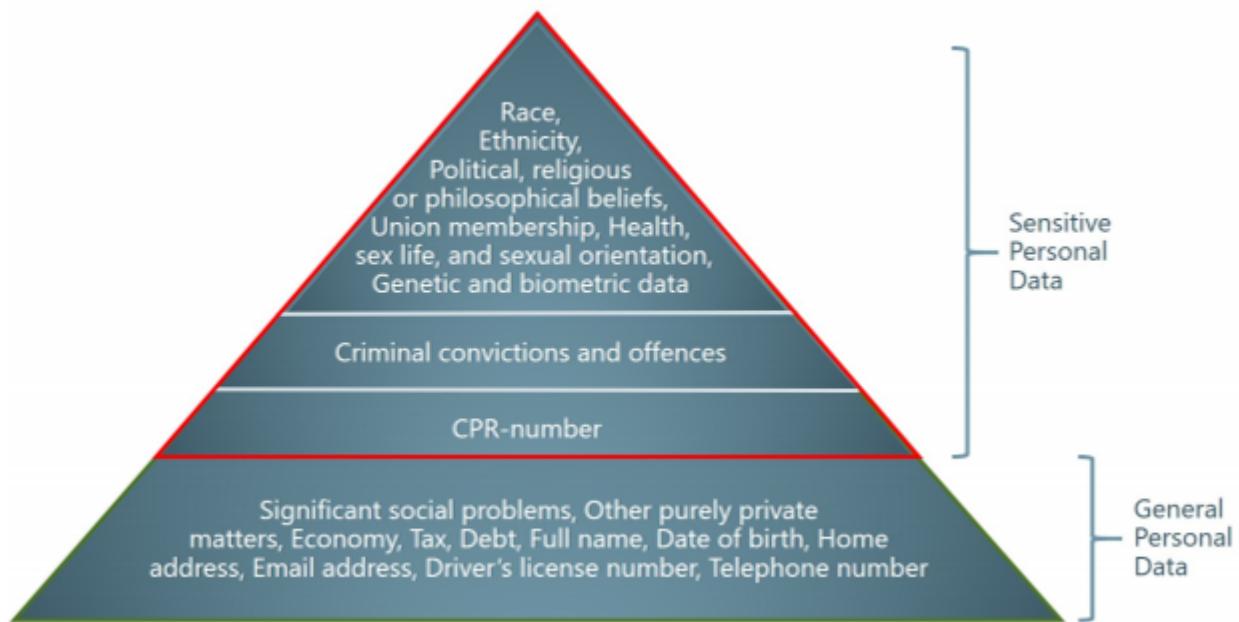
Normal – a change to production system, which requires approval*

Standard – a change to production system that can be implemented without further approval*

Emergency – change to production system, which requires immediate action and approval (afterwards)*

*) See "EG SOP – Change"

7.7.3.2 Classification of data:



Data handled in the feature or change shall be specified according to EG´s classification of data defined as:

Classification	Description	Example
1 Public	<p>Non-sensitive data that everyone can access upon request.</p> <p>Data requires no special protection.</p>	Data on public websites
2 Internal	<p>Internal data that can only be used and communicated internally in the company.</p> <p>Data requires some protection, and only approved people can access it.</p>	Meeting notes and invoices
3 Confidential	<p>Confidential data can only be used and shared by trusted people.</p> <p>Data requires special protection, e.g. encryption in connection with data transfer.</p>	Contacts and financial ledgers

Classification	Description	Example
4 PII - Person Identifiable Information - Normal	<p>Data within this category is subject to extra requirements as described in legislation.</p> <p>Access must be limited to few people with a documented work-related need.</p> <p>Data requires a high level of protection.</p>	Socialsecurity numbers, name, adress, telephone, email
5 PII - Person Identifiable Information - Sensitive	<p>Data within this category is subject to extra requirements as desribped in legislation.</p> <p>Access must be limited to very few people with a thoroughly documented work-related need.</p> <p>Data requires an extra high level of protection.</p>	Information about religion, politic alliance, sexuality, convictions, illnessess, salaries, personaldata regarding children. In general all information that can result in marginalization of the data subject.

Classification 4 and 5 is defined as "GDPR sensitive data" and requires special or extra special protection.

7.7.3.3 Threat classification

Threats imposed by the feature or change shall be analysed according to EG's Assets and threats in the category of IT Service defined as:

1. No threat
 - No risk at all or the probability *impact (risk factor) of foreseeable risk has a risk factor at 5 or below.
2. User error (Brugerfejl)
 - Unintentional user actions, improper handling of assets, lack of user education
3. Cyber terror attack (cyberangreb)
 - Destructive hacking, information tampering, website defacing
 - Denial of service attack, botnet attack, spam attack
 - State-sponsored attacks (APT)
 - Falsifying information, defacing of website
 - Hacking, espionage, eavesdropping, information interception
4. Penalties for non-compliance (GDPR non-compliance)
 - Violation of intellectual property rights
 - The company does not comply with relevant compliance requirements

- Incorrect publication of personal data
 - Failure to consider security during projects
5. Information leakage (Informationslæk)
- Excess user rights and privileges
 - Misconfiguration of access rights and security systems
 - Wrongful publication of data, embedding of hidden data in datafiles
 - Loss of portable information assets / data media
6. Deliberate misuse (Misbrug af bruger)
- Falsification, fraud, embezzlement
 - Unauthorized use, abuse of rights
 - Violation of intellectual property rights
7. Malicious code attack (Ondskabsfuld kodeangreb)
- Virus, worms, trojans, logical bombs, bots, root kits
8. Software error (Softwarefejl)
- Software failure, bugs, database corruption
 - Unauthorised or un-tested code
9. Maintenance or operations error (Vedligeholdelses- eller driftsfejl)
- Operations, support or maintenance staff error

7.7.3.4 Threat risk:

Threat Risk shall be specified as:

1. None
 - If threat classification is "No threat"
2. Low
 - if risk factor of the worst specified threat classification is 6 to 10
3. Medium
 - if risk factor the worst specified threat classification is 12 to 16
4. High
 - if risk factor of the worst specified threat classification is 20 to 25

Risk factor = probability * impact

Probability	Impact				
XX	Very low (1)	Low (2)	Medium (3)	High (4)	Very high (5)

Probability	Impact				
Very low (1)	1	3	3	4	5
Low (2)	2	4	6	8	10
Medium (3)	3	6	9	12	15
High (4)	4	8	12	16	20
Very high (5)	5	10	15	20	25

7.7.3.5 Risk analysis description

Use the following template for describing how threats are minimized by design, test or at deploying:

The risk of <specify threat> has a probability of <specify 1-5> and an impact of <specify 1-5>, which is reduced by <describe how this risk is reduced by design, test or at deploying>. <repeat for next threat>

Example:

The risk of unintentionally deleting personal data due to a user error has a probability of 5 and an impact of 4, which has been reduced by forcing the user to confirm deletion before the deletion happens. All deletions are logged with date, time and user initials in order to support tracking and restoring deleted data from backup.

7.7.3.6 Privacy by design (Privatlvsbeskyttelse af data)

Select applicable requirements to consider in order to comply with privacy by design:

1. Collect Consent Statement.
 - Save documentation for collected consent statement, which data, for which causes, and for how long.
Consent must be clear and distinguishable from other matters and provided in an intelligible and easily accessible form, using clear and plain language

- See Article 7 - Conditions for consent⁷⁶

2. Save For What Causes Data Are Saved

- Why do we save the data?
- Is it the system's responsibility to communicate this to the data subject?
- See Article 12 - Transparent information, communication and modalities for the exercise of the rights of the data subject⁷⁷
- See Article 13 - Information to be provided where personal data are collected from the data subject⁷⁸

3. Save Privacy Data for Limited Period

- Data is not allowed to be saved for more time than is necessary to solve the "Save For What Causes Data Are Saved"
- See Article 12 - Transparent information, communication and modalities for the exercise of the rights of the data subject⁷⁹
- See Article 13 - Information to be provided where personal data are collected from the data subject⁸⁰

4. Privacy Data Listing

- Listing on request. List all privacy data collected on request. Information to be provided where personal data are collected from the data subject
- See Article 15 - Right of access by the data subject⁸¹

5. Privacy Data Correction

- Correction on request. The data subject shall have the right to obtain from the controller without undue delay the rectification of inaccurate personal data concerning him or her. Taking into account the purposes of the processing, the data subject shall have the right to have incomplete personal data completed, including by means of providing a supplementary statement.
- See Article 16 - Right to rectification⁸²

6. Privacy Data Portability

- Export data on request (Portability). The right for a data subject to receive the personal data concerning them, which they have previously provided in a 'commonly used and machine readable format' and have the right to transmit that data to another controller
- See Article 20 - Right to data portability⁸³

7. Privacy Data Deletion

⁷⁶ <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

⁷⁷ <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

⁷⁸ <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

⁷⁹ <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

⁸⁰ <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

⁸¹ <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

⁸² <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

⁸³ <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

- Data deletion or anonymization on request. The software should include functionality (or a process) to delete or anonymize requested privacy data. The right to be forgotten entitles the data subject to have the data controller erase his/her personal data, cease further dissemination of the data, and potentially have third parties halt processing of the data. Depending on the backup and restore setup of the software, you may also need a process to store deletion requests such that you can perform them again in case of restore of previous data. Use a risk-based assessment in determining the need to store GDPR deletion requests outside the main environment, and applying those requests to any restore processes. In severe cases, you may need to have a process to apply deletions to the stored backups.
This should not be the case normally. But proper handling of backups (and other copies) of the data and a regular process to delete these must exist.
- See Article 17 - Right to erasure ('right to be forgotten')⁸⁴

8. Minimize Number of Data

9. Minimize Period

7.7.3.7 Access control (Adgang til data)

Select applicable requirements to consider in order to comply with access control:

1. Role-based access
 - Access to privacy data has to be granted on a strictly and documented work related need, and for no longer than absolutely necessary, e.g. by supporting user or role-based access rights.
2. Handle Work Related Access Documentation
 - The user must have workrelated reason to be able to access data
3. Log Granted Access
 - Log-in. It is mandatory to log any successful or unsuccessful attempt to log-in. Minimum of fields to be logged are:
 - i. "System"
 - ii. "User ID"
 - iii. "Time"
 - iv. "Successful" or "Unsuccessful"
4. Log Searches
 - It is mandatory to log all searches in tables containing privacy data. Do not log each field, that is accessed. Minimum of fields to be logged are:
 - i. "System"
 - ii. "User ID"
 - iii. "Time"

⁸⁴ <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504>

- iv. "Table"
- v. "Selection Criteria"
- vi. "types of privacy data" ("types of privacy data" means that the log levels should be:
Contact information, payroll information, Health information)

5. Log Views

- It is mandatory to log any activity, where a person have access to view privacy data. Minimum of fields to be logged are:
 - i. "System"
 - ii. "User ID"
 - iii. "Time"
 - iv. "Table"
 - v. "all viewed privacy fields"

6. Log Changes

- It is mandatory to log any activity, where a person have changed privacy data. Minimum of fields to be logged are:
 - i. "System"
 - ii. "User ID"
 - iii. "Time"
 - iv. "Table"
 - v. "all changed privacy fields"
 - vi. "old value", "new value"

7. Log Deletions.

- a. It is mandatory to log any activity, where a person have deleted or anonymised privacy data.
Minimum of fields to be logged are:
 - "System"
 - "User ID"
 - "Time"
 - "Table"
 - "all deleted privacy fields"

General Logging Requirements:

1. If nothing else specified in the 'EG Corporate Privacy Policy' or in the data processor agreements with your customers, log files are to be saved for 6 months.
Access to log files must be granted to only a very limited number of people, and it must be protected against changes and deletions.
2. You should remove GDPR Sensitive data from common program logs, or as a minimum anonymize or encrypt the logs if GDPR Sensitive data is included.
3. Make sure the log can be accessed for usage in GDPR audits.
4. Consider the need for SQL database audits (SQL 2012+) if multiple sources access the database.
Consider implementing these audits just for statements run by users

outside the application environment. Determine the need for this by making a risk based assessment.

Note: It is mandatory to log all the above mentioned activities in tables containing privacy data, whether it is from an application or on a database level.

7.7.3.8 Data protection and encryption (Beskyttelse og kryptering af data)

Select applicable requirements to consider in order to comply with data protection related to feature or change:

1. Handle data in storage
 - Data at data classification 4 and 5 ("GDPR sensitive data") shall be encrypted during storage if user with no work related need or unauthorized user can get access to the data.
2. Handle data in use
 - Data at data classification 3 or above shall be considered protected in use, e.g. by hiding password characters entered in password field.
3. Handle data in transport
 - Data at data classification 2 or above shall be encrypted during transport and data at data classification 4 and 5 requires following integration rules:
 - i. IC 1 - Data generated by our software. GDPR Sensitive Data generated by an EG product in a specific integration format. Customers handle all transport and further actions with the data (I.e. datafiles, views, data dumps).
 1. Functionality must be specified in the delivery contract and data processor agreement with the customer.
 - ii. IC 2 - Data transported by our software. GDPR Sensitive Data generated by an EG product which is also responsible for transport to external system or service (I.e. webservices, cloud services).
 1. This type of integration must be mentioned in the delivery contract with the customer, and may require mentioning in the data processor agreement appendix, depending on the type of GDPR data being trans-ported.
 2. The customer may be required to enter a separate data processor agreement with the third party, and should be informed about this requirement.
 3. Also consider the steps related to Encryption at motion.
 - iii. IC 3 - Tight Data integrations. GDPR Sensitive Data generated or handled by an EG product which is tightly coupled with an external system (I.e. 2-way integrations, components, or solutions sold together as a package).
 1. This type of integration must be mentioned in the data processor agreement appendix.
 2. The customer is most likely required to enter a separate data processor agreement with the third party, and should be informed about this requirement.
 3. EG probably also need to have a sub data processor agreement with the subcontractor – see 'SOP Sub Data Processor Agreement' and use template.
 4. Also consider the steps related to Encryption at motion.

7.8 7.8 - Jira & Confluence permissions schema

7.8.1 EG Jira

EG's Jira <https://jira.eg.dk/> default NGA permissions scheme consists of five roles, which define the permission sets in scope of Jira access for all users:

- **Administrator** - dedicated for a person or very narrow group in every Jira product, allowing limited administration of the Jira project; contains all permissions of the other roles
- **Observer** - dedicated for Jira project externals, who need read-only access to the project with no other permissions than browsing
- **Scrum Master** - based on the Developer role but extended with additional permissions allowing management of sprints, such as: creation, deletion, starting and completing, editing, etc.
- **Product Owner** - based on the Developer role but extended with additional permissions allowing completion of epics, user stories and bugs; also allowing deletion of all issue types in Jira
- **Developer** - standard permission set dedicated for Developers, including Software Engineers, Testers, UX designers, Consultants, etc. Contains all necessary rights to collaborate in the Jira project
- **External Observer** - dedicated for EG external users with Observer role
- **External Developer** - dedicated for EG external users with Developer role
- **Delete** - dedicated for any user of the Jira project who should have the permission to permanently delete Jira issues within their project

The complete, detailed table with permission schema / role mappings:

Project permissions		Granted to
Permission	Project role	
Administer Projects Ability to administer a project in Jira. <input checked="" type="checkbox"/> Extended project administration Grant extended project administration permissions.	Administrator	
Browse Project Archive Ability to browse archived issues from a specific project.	Administrator Product Owner	
Browse Projects Ability to browse projects and the issues within them.	Observer Administrator Product Owner Scrum Master Developer External observer External developer	
Edit Sprints Ability to edit sprint name and goal.	Scrum Master Administrator	
Manage Sprints Ability to manage sprints.	Scrum Master Administrator	
Start/Complete Sprints Ability to start and complete sprints.	Scrum Master Administrator	
View Development Tools Allows users in a software project to view development-related information on the issue, such as commits, reviews and build information.	Developer Administrator External developer	
View Read-Only Workflow Users with this permission may view a read-only version of a workflow.	Administrator	

Issue permissions

Permission	Granted to
Archive Issues Ability to archive issues for a specific project.	Project role <ul style="list-style-type: none">AdministratorProduct Owner
Assignable User Users with this permission may be assigned to issues.	Project role <ul style="list-style-type: none">AdministratorProduct OwnerScrum MasterDeveloperExternal developer
Assign Issues Ability to assign issues to other people.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer
Close Issues Ability to close issues. Often useful where your developers resolve issues, and a QA department closes them.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer
Create Issues Ability to create issues.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer
Delete Issues Ability to delete issues.	Group <ul style="list-style-type: none">jira-administrators
Edit Issues Ability to edit issues.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer
Link Issues Ability to link issues together and create linked issues. Only useful if issue linking is turned on.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer
Modify Reporter Ability to modify the reporter when creating or editing an issue.	Project role <ul style="list-style-type: none">AdministratorProduct OwnerScrum MasterDeveloper
Move Issues Ability to move issues between projects or between workflows of the same project (if applicable). Note the user can only move issues to a project he or she has the create permission for.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer
Resolve Issues Ability to resolve and reopen issues. This includes the ability to set a fix version.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer
Restore Issues Ability to restore issues for a specific project.	Project role <ul style="list-style-type: none">AdministratorProduct Owner
Schedule Issues Ability to view or edit an issue's due date.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer
Set Issue Security Ability to set the level of security on an issue so that only people in that security level can see the issue.	Project role <ul style="list-style-type: none">Administrator Group <ul style="list-style-type: none">USR_Atlassian_EG_GDPR_Compliance
Transition Issues Ability to transition issues.	Project role <ul style="list-style-type: none">AdministratorDeveloperProduct OwnerScrum MasterExternal developer Group <ul style="list-style-type: none">jira-software-bulk

Voters & watchers permissions

Permission	Granted to
Manage Watchers Ability to manage the watchers of an issue.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer
View Voters and Watchers Ability to view the voters and watchers of an issue.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer

Comments permissions

Permission	Granted to
Add Comments Ability to comment on issues.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer Observer
Delete All Comments Ability to delete all comments made on issues.	Project role <ul style="list-style-type: none"> Administrator
Delete Own Comments Ability to delete own comments made on issues.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer
Edit All Comments Ability to edit all comments made on issues.	Project role <ul style="list-style-type: none"> Administrator
Edit Own Comments Ability to edit own comments made on issues.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer

Attachments permissions

Permission	Granted to
Create Attachments Users with this permission may create attachments.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer
Delete All Attachments Users with this permission may delete all attachments.	Project role <ul style="list-style-type: none"> Administrator
Delete Own Attachments Users with this permission may delete own attachments.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer

Time tracking permissions

Permission	Granted to
Delete All Worklogs Ability to delete all worklogs made on issues.	Project role <ul style="list-style-type: none"> Administrator
Delete Own Worklogs Ability to delete own worklogs made on issues.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer
Edit All Worklogs Ability to edit all worklogs made on issues.	Project role <ul style="list-style-type: none"> Administrator
Edit Own Worklogs Ability to edit own worklogs made on issues.	Project role <ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer

Work On Issues	Project role
Ability to log work done against an issue. Only useful if Time Tracking is turned on.	<ul style="list-style-type: none"> Administrator Developer Product Owner Scrum Master External developer

7.8.2 EG Confluence

EG's Confluence <https://confluence.eg.dk> defines a set of permissions which are assignable on a per-person or per-group basis. The following permission scheme is followed in EG for the mentioned roles:

Permission \ Role	Developers & Scrum Master	Product Owner & Manager	BU Manager
View All	✓	✓	✓
Add Pages	✓	✓	✓
Delete Pages	✗	✓	✓
Add Comments	✓	✓	✓
Delete Comments	✗	✓	✓
Add Attachments	✓	✓	✓
Delete Attachments	✗	✓	✓
Add/Delete Restrictions	✗	✗	✓
Space Admin	✗	✗	discussed individually

7.9 7.9 - ServiceNow integration

- 7.9.1 - Jira - ServiceNow integration board and issue flow (see page 377)
- 7.9.2 - Jira - ServiceNow integration fields mapping (see page 380)
- 7.9.3 - Jira - ServiceNow integration communication via comments (see page 384)

7.9.1 7.9.1 - Jira - ServiceNow integration board and issue flow

In this chapter, you will get the information about handling the issues coming to your Jira project from ServiceNow.

Integration with ServiceNow will result in changes in your Jira project outlook. This enables proper issues grouping and handling in our Jira project. The changes are introduced in the following areas:

- dedicated "[Project key] ServiceNow Board" is created
- "servicenow_new" label is being added to your labels list
- "ServiceNow" user is created

7.9.1.1 ServiceNow dedicated Jira board

To distinguish all issues that are originally coming from the ServiceNow to our Jira project, a new dedicated board has been created.

"[Project key] ServiceNow Board" is available in the board list on the left side menu. When we go to this board we will be able to see all the issues that have come from ServiceNow and have not been evaluated yet. All issues coming from ServiceNow have a "ServiceNow" user as a reporter and are labeled with the "servicenow_new" label. If the supporter from ServiceNow updated the team name, it will also be addressed after migration to Jira.

The screenshot shows the Jira interface with the "CLN | ServiceNow Board" selected in the sidebar. The backlog section lists several issues under "CLN Support S1". One specific issue, CLN-14642, is highlighted and shown in a detailed view on the right. The issue card includes fields such as Estimate (Unestimated), Remaining (Unestimated), Sprint (CLN Support S1), Team name (Team Groot), Reporter (ServiceNow), Assignee (Unassigned), Product Item (None), Issue category (Maintenance and defects), Status (IN PROGRESS), Priority (Low), Component/s (None), Labels (servicenow_new), Affects Version/s (None), and Fix Version/s (None). The "servicenow_new" label is circled in red.

From this point of view, Developers can review, and evaluate the newly introduced issues. Developers in teams or individually can validate if the tickets are having all the needed input, initial parameters and if there are assigned to the proper team. If some input or information is missing it can be commented on with the request for an update. All the comments added in this stage will be also visible in the ServiceNow, so consultants can update the requested content before moving forward.

When the evaluation is done (including the input, priority, and parameters) Product Owner can prioritize and distribute the issues to be managed, according to the Agile process established in your Business Unit. The selected issues can be later prioritized in the product backlog along with other items or forecasted for one of the upcoming sprints immediately. Items with the highest priority can be considered by the agile team to be added to the ongoing sprint. KanBan teams prioritize the evaluated items in the backlog similarly - once approved for development, they should appear on the KanBan board to be handled with respect to the overall priority of product items in development. By adding/updating the proper team name the issues will be also visible on your dedicated team board. From this moment you can manage the tickets on a team level, including continuing the refinement process, sprint assignment, and completion of the issue.

The screenshot shows a Jira Kanban board titled "Kanban board". The board has three columns: "TO DO" (4 issues), "IN PROGRESS" (49 issues), and "DONE" (14 of 44 issues). The "DONE" column has a "Release..." button. The "IN PROGRESS" column has a "Board" dropdown and a refresh icon. The sidebar on the left includes a "Create board" button with a mouse cursor hovering over it, and sections for "BOARDS IN THIS PROJECT" (Kanban board), "PROJECT SHORTCUTS" (Issues, Components), and "+ Add link".

To provide the proper flow of the information between ServiceNow and Jira, dedicated fields and statuses are being instantly integrated between the tools. More about integrated fields, statuses, and information can be found in the [7.9.2 - Jira - ServiceNow integration fields mapping \(see page 380\)](#) chapter.

The below table is presenting the information about the visibility of the issues in the particular Project boards:

Jira issues parameters	Project Main Board	Team dedicated board (In the setup with multiple teams)	ServiceNow board
"servicenow_new" label present and "Team name" is empty	X	X	V
"servicenow_new" label present and "Team name" is updated	X	V	V
"servicenow_new" label is not present and "Team name" is empty	V	X	X

"servicenow_new" label is not present and "Team name" is updated	V	V	X
---	---	---	---

7.9.1.2 "servicenow_new" label

All the issues coming from ServiceNow are initially labeled with the "servicenow_new" label. Under this label, the dedicated "[Project key] ServiceNow Board" is configured, so if the label is removed, the ticket won't be visible on the board (as per the above table).

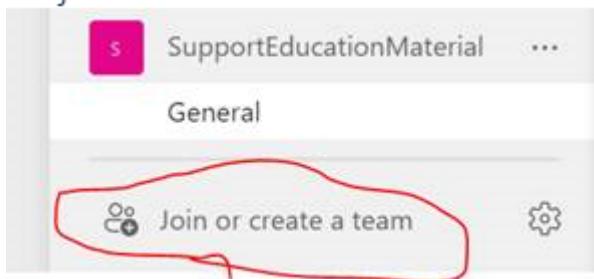
Priority: ▼ Low
 Component/s: None
 Labels: servicenow_new
 Affects Version/s: None
 Fix Version/s: None

7.9.1.3 Dedicated materials about ServiceNow - Jira integration

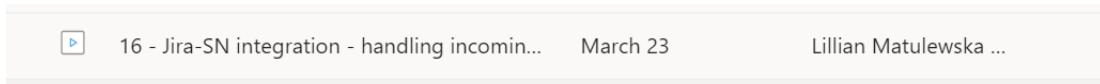
The detailed materials about ServiceNow and handling issues from ServiceNow in Jira can be found in MS Teams "SupportEducationMaterial" channel.

The channel can be found under Teams > SupportEducationMaterial > Documents > General > ServiceNow > English > Videos for education

If you can't see this channel in MS Teams please use the "Join or create a team" function and find and join the channel first. You can find there also additional materials about ServiceNow if needed.



After joining the above team in MS Teams we strongly advise you to review video No. 16:



7.9.2 7.9.2 - Jira - ServiceNow integration fields mapping

This chapter illustrates the dependency between Jira and ServiceNow in the scope of the integration between the two systems allowing for seamless work by the supporters and developers, on those issues originating from ServiceNow.

7.9.2.1 Integration scope

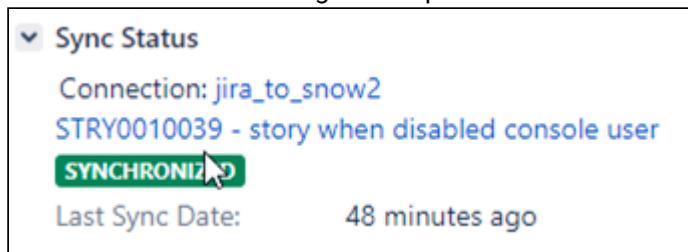
The following table illustrates what issue types created in ServiceNow are mapped onto issue types created in Jira, based on their **Classification** field in ServiceNow.

ServiceNow	Jira
Incident	Bug
Service Request - Information	Task
Service Request - Request	Task
Feature	Story

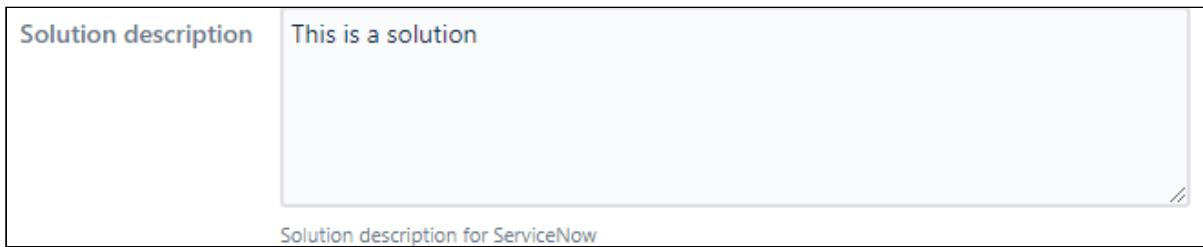
7.9.2.2 Changes in Jira

To support the integration, 3 new elements have been enabled in Jira with focus on the process flow.

- **ServiceNow ID & Sync status** - show the synchronization status between the Jira issue and ServiceNow ticket including the unique ServiceNow ID and a link to ServiceNow Story.



- **Solution Description text field** - an additional text field for defining and describing the solution to the incident/request stated in the ticket. The Solution Description will be synchronized to **Closing Notes** field in ServiceNow:



- **Default ServiceNow reporter** - allows distinguishing issues in Jira which are originating from ServiceNow; all issues coming from ServiceNow have the **ServiceNow** user set as the reporter:

The image shows a dropdown menu for the reporter field in Jira. The 'Unassigned' option is selected. Other options include 'Assign to me' and 'ServiceNow'.

- **Assignee** - by default, all issues created in Jira are assigned to the **ServiceNow** user. If the issue in Jira gets reassigned to a developer for example and **there are changes on the corresponding ServiceNow issue**, issue in Jira gets **unassigned**.
- **Contact name** - an additional text field for defining SN ticket reporter. Field is read only.
- **Contact number** - an additional text field for defining SN contact number. Field is read only.
- **SN Priority** - an additional text field for defining SN Priority. Field is read only.
- **SLA Breach Date** - an additional text field for defining SLA. Field is read only.
- **Account** -
- **Master data number** -
- **SN Assignment group** - an additional text field for defining SN Assignment group. Field is read only.
- **SN Assignee** - an additional text field for defining SN ticket assignee. Field is read only.
- **SN Story** - an additional text field for defining SN ticket. Field is read only.
- **SN Tickets Count** -
- **SN Ticket** - an additional text field for defining SN Ticket. Field is read only.

7.9.2.3 Issue field value synchronization

Once an issue is created in Jira, some of the fields are being subject to synchronization with ServiceNow when data is modified or new data is entered.

The **External Reference** field in ServiceNow gets updated with Jira **issue key**, when the story is initially synchronized with Jira.

The following table provides a listing of the synchronized fields in Jira along with their ServiceNow counterparts and a definition of the sync. direction with 3 possibilities:

- only changes in ServiceNow are propagated to Jira and not from Jira to ServiceNow (SN → Jira)
- only changes in Jira are propagated to Jira and not from ServiceNow to Jira (Jira → SN)
- changes are propagated in both directions (SN ↔ Jira)

ServiceNow	Jira	Synchronization	Comment
u_jira_project	Jira Project	SN → Jira	Jira project key, like CLN or ESE .
u_jira_team	Team Name	SN ↔ Jira	i.e. Team Groot
u_external_reference	Key	SN ← Jira	Jira issue key, like CLN-2322 . Jira issue key is synced back to ServiceNow as External Reference.
classification	Type	SN ↔ Jira	Story, Bug or Task
short_description	Summary	only SN → Jira	
Internal description	Description	only SN → Jira	
state	Status	only Jira → SN	see status value mappings below
priority	Priority	SN ↔ Jira	see priority value mappings below
work_note	Comment	SN ↔ Jira	
attachment	Attachment	SN → Jira	attachments are synchronized one way only: ServiceNow → Jira.
u_activity_id	ERP Activity ID	SN → Jira	
close_notes	Solution Description	Jira → SN	
u_sprint	Sprint	Jira → SN	planned. Jira Sprint name synced to ServiceNow into Sprint field.
u_originating_record	part of issue Description	SN → Jira	the ServiceNow story originating record (case) is appended to Jira issue description

ServiceNow	Jira	Synchronization	Comment
u_related_case_count	part of issue Description	SN → Jira	the number of ServiceNow related cases is appended to Jira issue description

7.9.2.3.1 Status value mappings

ServiceNow issue type	ServiceNow	Jira
Incident/Request	New	To Do
	Assess	Ready
	Work In Progress	In Progress
	Closed	Done
Feature	Draft	To Do
	Ready	Ready
	Work In Progress	In Progress
	Ready For Testing	In Progress
	Testing	In Progress
	Complete	Done
	Cancelled	-

7.9.2.3.2 Priority value mappings

ServiceNow	Jira
Critical	Highest
High	High
Moderate	Medium
Low	Low
Planning	Lowest

7.9.3 7.9.3 - Jira - ServiceNow integration communication via comments

The Jira - ServiceNow integration enables developers and support consultants to communicate together via Jira comments, looking from a Jira user perspective.

There are 3 types of comments that can be distinguished in Jira with respect to ServiceNow integration. These types are indicated by the restriction that is set on the comment when adding it by a user.

- Customer restriction comment - will be sent to ServiceNow and will be visible to the Customer
- ServiceNow restriction comment - will be sent to ServiceNow but will not be visible to the Customer
- *Default (no restriction) or any other restriction comment* - will not be sent to ServiceNow and will be only visible in Jira comments

The screenshot shows the Jira interface with the 'Comments' tab selected. A rich text editor is open, and a dropdown menu titled 'Project Roles' is displayed, listing roles such as Administrator, Customer, Developer, Product Owner, Scrum Master, and ServiceNow.

By selecting the role restriction when adding a comment, the Jira user indicates the type of comment to be sent or not sent to ServiceNow.

In order for a Jira user to have the possibility of selecting a comment restriction type, the user needs to have assigned these roles in the Jira project **users & roles setting**. This can be done by any Jira project admin of that project.

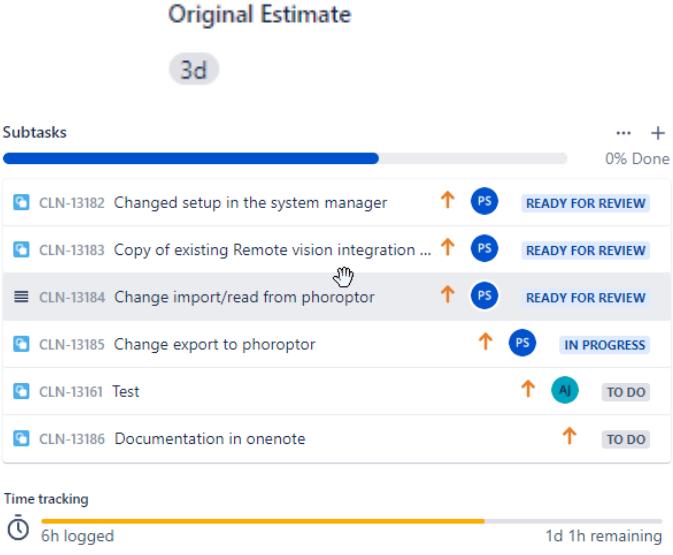
The screenshot shows the 'Users and roles' section of the Jira project settings. It lists several users with their assigned roles. A red box highlights a specific user entry where the 'Customer' role is selected.

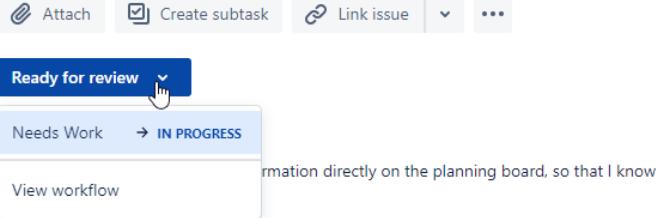
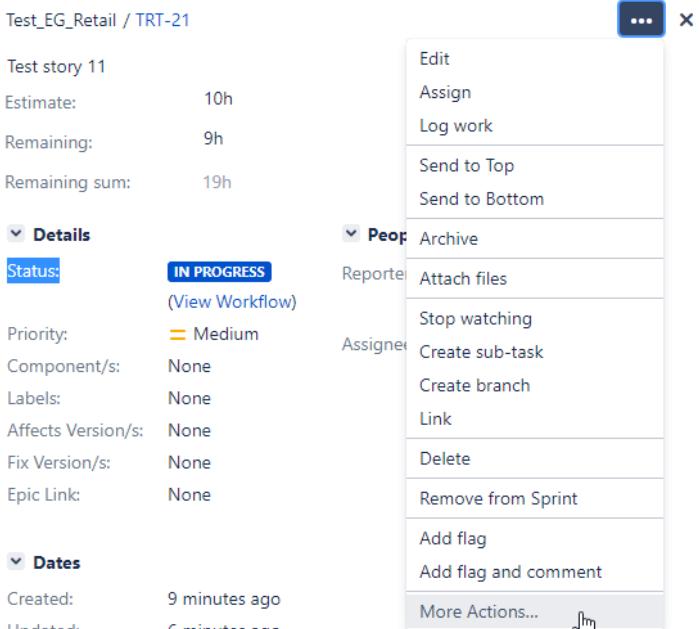
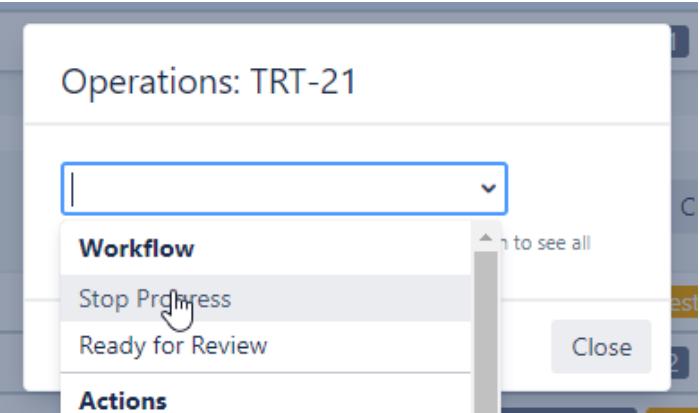
User	Role	Last Activity	Action
Multiple (3)	Administrator	1 hour ago	Remove
Multiple (3)	Customer	3 hours ago	Remove
	Developer	1 hour ago	Remove
	Customer	6 hours ago	Remove
	Customer	5 days ago 12:51 PM	Remove

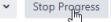
7.10 7.10 - [FAQ] Cloud to On-Prem migration

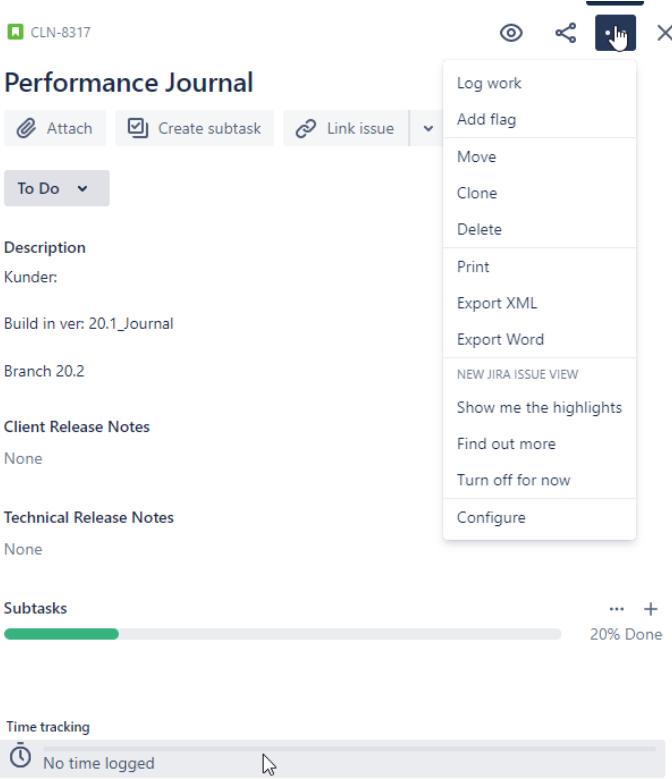
7.10.1 Jira

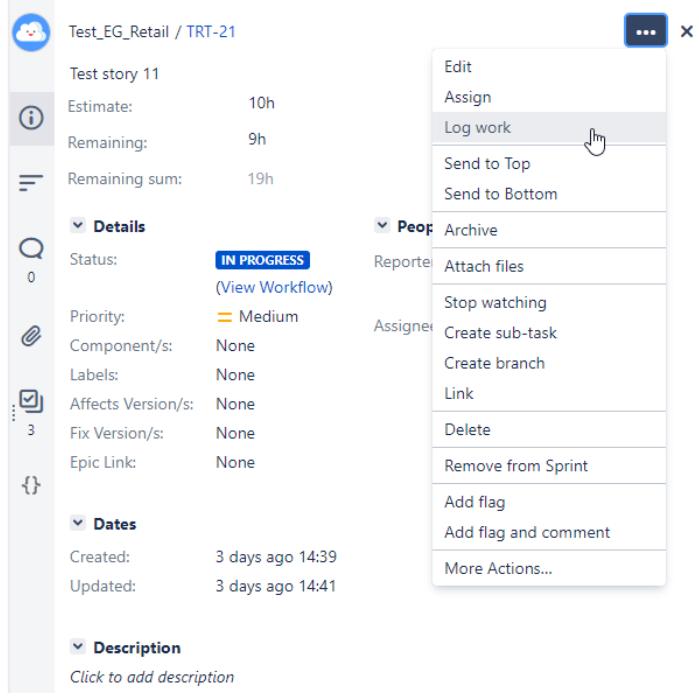
Question	Answer
<p>Where are the avatar filters on the Backlog view?</p> <p>Cloud:</p>  <p>On-Premise:</p> 	<p>The avatar filters representing team members involved in the project issues, which are used for filtering by assignee are only available in the Cloud version of Atlassian software and are not visible in Jira On-Premise.</p> <p><u>workaround:</u> it is possible to filter by assignee, simply by typing that person's username into the search text filter field.</p>
<p>Where are the avatar filters and search text filter on the Active sprints view?</p> <p>Cloud:</p>  <p>On-Premise:</p> 	<p>The avatar filters representing team members involved in the project issues, which are used for filtering by assignee, as well as the search text filter are only available in the Cloud version of Atlassian software and are not visible in Jira On-Premise.</p> <p><u>workaround:</u> while no direct workaround is available, it is still possible to define quick filters which will allow replicating the expected filtering result, such as: My Issues, Issues in Status X, etc.</p>

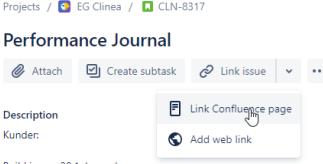
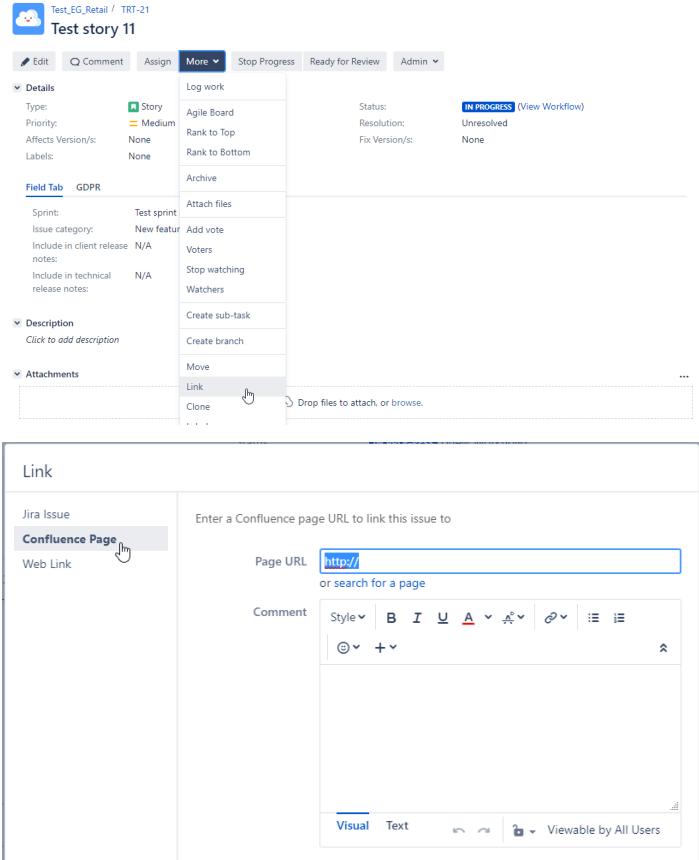
Question	Answer																														
<p>How did the presentation of remaining time estimates change on the issue view?</p> <p>Cloud:</p>  <p>The screenshot shows the Jira Cloud issue view for an issue with an original estimate of 3d. It includes a list of sub-tasks with their own estimates and status (e.g., READY FOR REVIEW, IN PROGRESS, TO DO). Below the sub-tasks is a time tracking section showing 6h logged and 1d 1h remaining.</p> <p>On-Premise:</p> <table border="0" data-bbox="282 1147 641 1316"> <tr> <td>Estimate:</td> <td>10h</td> </tr> <tr> <td>Remaining:</td> <td>9h</td> </tr> <tr> <td>Remaining sum:</td> <td>19h</td> </tr> </table> <p>Sub-Tasks:</p> <table border="0" data-bbox="181 1356 854 1567"> <tr> <td>TRT-22</td> <td>sub-task A</td> <td>TO DO</td> <td>3h</td> </tr> <tr> <td>TRT-23</td> <td>sub-task B</td> <td>TO DO</td> <td>2h</td> </tr> <tr> <td>TRT-24</td> <td>sub-task C</td> <td>TO DO</td> <td>5h</td> </tr> <tr> <td colspan="4">$\Sigma 10h$</td> </tr> </table> <p>Time Tracking:</p> <table border="0" data-bbox="181 1608 854 1792"> <tr> <td>Estimated:</td> <td>20h</td> </tr> <tr> <td>Remaining:</td> <td>19h</td> </tr> <tr> <td>Logged:</td> <td>1h</td> </tr> <tr> <td><input checked="" type="checkbox"/> Include sub-tasks</td> <td></td> </tr> </table>	Estimate:	10h	Remaining:	9h	Remaining sum:	19h	TRT-22	sub-task A	TO DO	3h	TRT-23	sub-task B	TO DO	2h	TRT-24	sub-task C	TO DO	5h	$\Sigma 10h$				Estimated:	20h	Remaining:	19h	Logged:	1h	<input checked="" type="checkbox"/> Include sub-tasks		<p>The presentation of the remaining time estimates on issue view is much more detailed and improved as compared to the Cloud version of the Atlassian software.</p> <ul style="list-style-type: none"> the estimations expressed throughout the system are always coherently represented by one selected metric (in our case hours) instead of converting to days on the right-side view of the issue the estimations expressed on the parent issue now present apart from the original estimate and remaining time, the remaining time including the underlying sub-tasks' estimations (3rd field - <i>remaining sum</i>) the sub-task listing on the right-side issue details view now presents individual estimates of the sub-tasks, as well as the sum of all sub-tasks' remaining time estimates the time tracking module looks a bit different and presents 3 variables for the issue, including the original estimate, remaining time estimate and logged time; the time tracking module also includes the possibility to present the values with or without including the sub-task estimations
Estimate:	10h																														
Remaining:	9h																														
Remaining sum:	19h																														
TRT-22	sub-task A	TO DO	3h																												
TRT-23	sub-task B	TO DO	2h																												
TRT-24	sub-task C	TO DO	5h																												
$\Sigma 10h$																															
Estimated:	20h																														
Remaining:	19h																														
Logged:	1h																														
<input checked="" type="checkbox"/> Include sub-tasks																															

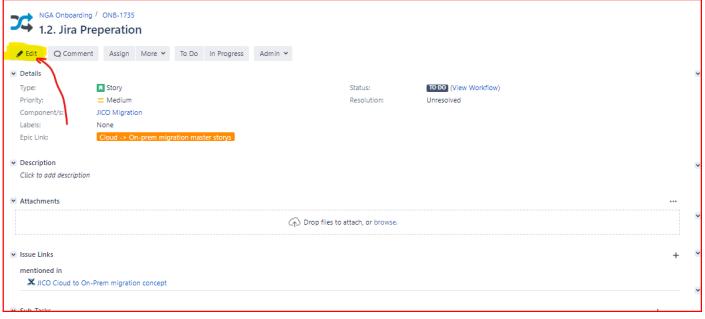
Question	Answer
<p>How do I change the status of an issue?</p> <p>Cloud:</p>  <p>On-Premise:</p>  	<p>The status change process in the On-Premise version of Atlassian software is a bit more diverse than that of the Cloud.</p> <p>The status of an issue, while in the right-side details view needs to be conducted by accessing “More actions...” option from the “...” menu and selecting the proper workflow transition. Jira On-Premise operates rather on transitions as compared to Cloud, where the user selects the proper target status.</p> <p>In full page view, the status of an issue can be changed by selecting the button from the top menu with the proper workflow transition, as compared to selecting the target status from a drop-down in the Cloud version of the software.</p>

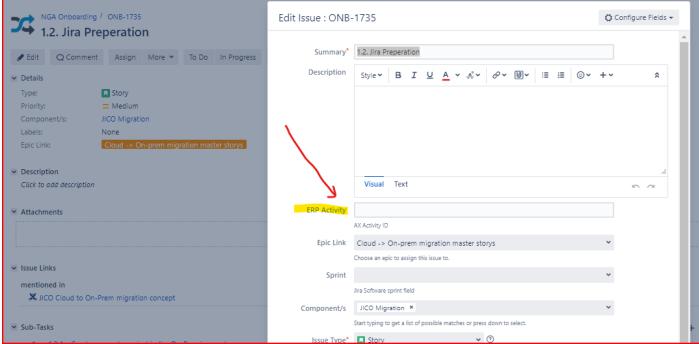
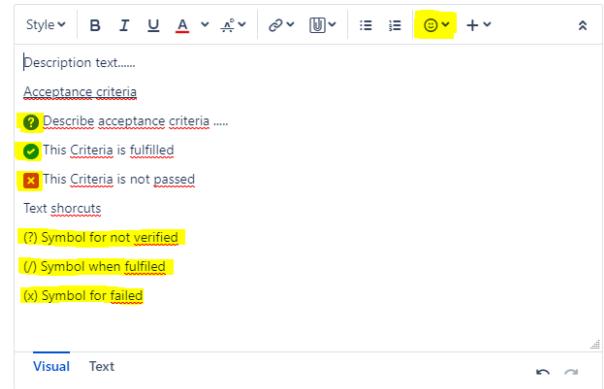
Question	Answer
 Test_EG_Retail / TRT-21 Test story 11  Edit  Comment  Assign  More  Stop Progress  Ready for Review  Admin  Details Type:  Story Resolution:  IN PROGRESS View Workflow Priority:  Medium Status:  Unresolved	

Question	Answer
<p>How can I now log my work?</p> <p>Cloud:</p>  <p>The screenshot shows a Jira Cloud issue page for CLN-8317 titled "Performance Journal". The "Log work" option is highlighted in the context menu.</p> <p>On-Premise:</p>	<p>Logging time in Jira On-Premise is very similar to that functionality of Jira Cloud. Fundamentally it is possible to log work in both Jira Cloud and On-Premise by selecting such action from the "..." menu in the upper right corner of the right-side view.</p> <p>In Jira Cloud it is also possible to use the time tracking module in the view. Jira On-Premise allows to log work through the different looking time tracking module, however only in the full page view of the issue, where a user may add worklogs by clicking "+" next to the module.</p>

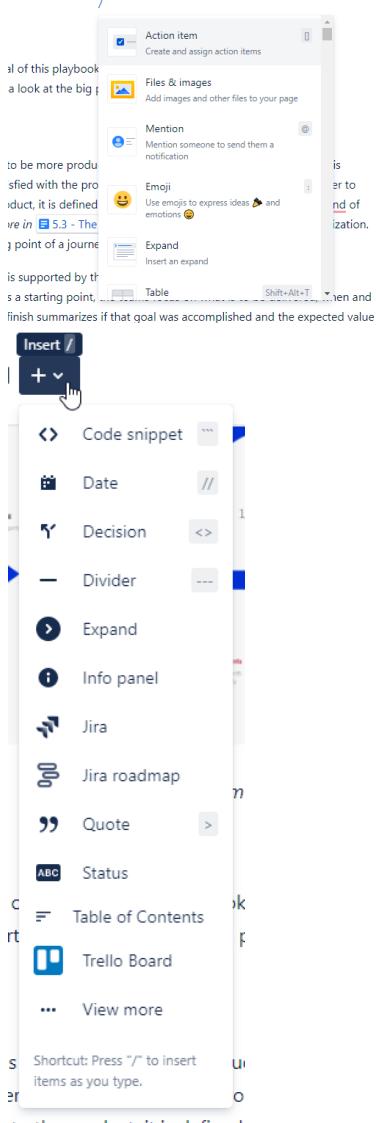
Question	Answer
 <p>Test_EG_Retail / TRT-21</p> <p>Test story 11</p> <p>Estimate: 10h Remaining: 9h Remaining sum: 19h</p> <p>Details</p> <ul style="list-style-type: none"> Status: IN PROGRESS (View Workflow) Priority: Medium Component/s: None Labels: None Affects Version/s: None Fix Version/s: None Epic Link: None <p>Dates</p> <ul style="list-style-type: none"> Created: 3 days ago 14:39 Updated: 3 days ago 14:41 <p>Description</p> <p>Click to add description</p> <p>More Actions...</p> <p>Log work</p> <p>People</p> <p>Dates</p> <p>Time Tracking</p>	

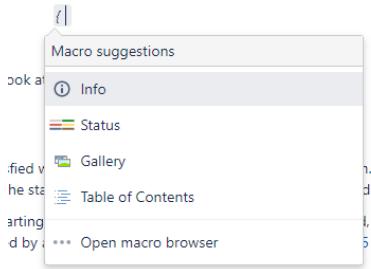
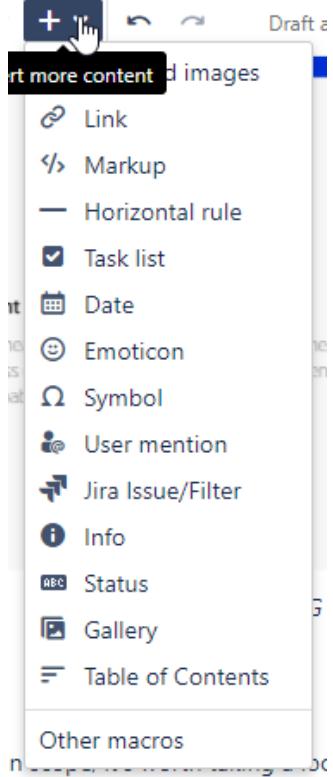
Question	Answer
<p>How can I now link a Confluence page to a Jira issue? How do I link another Jira issue?</p> <p>Cloud:</p>  <p>On-Premise:</p> 	<p>Linking a Confluence page to a Jira issue in Cloud was a dedicated option in the drop-down menu next to the "Link issue" button. In Jira On-Premise, this is an integrated part of the general linking functionality of Jira issues.</p> <p>In order to link a Confluence page to a Jira issue, a user must select the "Link" option from the "More" drop-down menu, and then select the link type Confluence Page. The same way by selecting the Jira Issue link type, a user may connect two or more existing Jira issues together. This was previously accessible through a dedicated button on the issue in Jira Cloud.</p>

Question	Answer
<p>Where are some of the fields (such as ERT Activity) and how do I define values for them on an existing issue?</p> <p>Labels None</p> <p>Original Estimate 0m</p> <p>Time tracking  4h logged</p> <p>Issue category Technical debt</p> <p>ERP Activity None</p> <p>Epic Link Venteliste videreudvikling</p> 	<p>In Jira cloud all of the fields were visible in full screen and board side-panel view.</p> <p>In Jira on-prem only fields with defined values are presented in the full screen view; if the field is not there it means the value is empty. In order to enter a value for one of these fields (like ERP Activity) you need to access the Edit mode by clicking the Edit button and entering a value into the proper field.</p> <p>On the board in the right-side panel view however, the fields are accessible and visible even if empty, so the value can be entered there without going into edit mode.</p>

Question	Answer
 <p>The screenshot shows the Jira interface. On the left, there's a sidebar with project navigation and a 'Backlog' section. In the center, an 'Edit Issue' dialog is open for issue ONB-1735, specifically for the '1.2. Jira Preparation' epic. The 'ERP Activity' field in the dialog has a red arrow pointing to the 'ERP Activity' dropdown in the backlog card on the right. The backlog card shows a card for 'test 1' with various details like estimate, story points, and status.</p>	
<h3>How do I format acceptance criteria, without the use Info-Panels</h3>  <p>The screenshot shows the Jira 'Description' editor. It contains acceptance criteria formatted using rich text tools. Symbols like question marks and checkmarks are highlighted with yellow boxes. The text includes descriptions like 'Acceptance criteria', 'Describe acceptance criteria', 'This Criteria is fulfilled', 'This Criteria is not passed', and 'Text shortcuts' with symbols for 'Symbol for not verified', 'Symbol when fulfilled', and 'Symbol for failed'.</p>	<p>Infopanel is currently not available with Jira on-server. A workaround is to use the symbols shown on the image to indicate the state of the acceptance criteria.</p>

7.10.2 Confluence

Question	Answer
<p>How do I add macros/items into Confluence pages?</p> <p>Cloud:</p>  <p>On-Premise:</p>	<p>Both Confluence Cloud and On-Premise allow inserting macros / items of various type into the pages to enrich the content. This can be done in both cases through the classic selection of the "+" drop-down and choice of the desired feature.</p> <p>The "/" shortcut does not apply in case of Confluence On-Premise, as in the Cloud version, however now the "{" can be used in order to select one of the quick macros or pick one from the macro browser.</p>

Question	Answer
 	

7.11 7.11 - Jira Advanced Roadmaps

7.11.1 What is Jira Advanced Roadmaps?

Jira Advanced Roadmaps (JAR, Plans) is a powerful solution for big-picture planning. Available as part of the Atlassian Jira Software Data Center. It allows you to combine issues from boards, projects, and filters to

create an all-encompassing plan that spans multiple teams or Jira projects. The benefit of using a plan is that it establishes the bigger picture and shows how a team's work relates to the work of other teams. It visualizes how each team and other participants contribute to broad common or shared goals.

Jira Advanced Roadmaps can be very useful in:

1. **Cross-Team Collaboration and Planning:** It facilitates collaboration among multiple teams by providing a common view of work items and dependencies, allowing for better planning, coordination, and communication.
2. **Dependency Management:** You can identify and manage dependencies between different projects, epics, and stories. This helps in preventing bottlenecks and delays caused by interrelated work items.
3. **Progress Tracking:** Teams can track the progress of epics, features, and stories in real time. This visibility allows for better decision-making and adjustment of plans as needed.
4. **Scenario Planning:** Advanced Roadmaps allow you to create different planning scenarios to explore what-if scenarios. This is helpful in assessing the impact of changes or new work items on the overall project/feature timeline.
5. **Release Planning:** It assists in creating and managing release plans, ensuring that the right features and stories are prioritized for upcoming releases.
6. **Capacity Planning:** Teams can allocate resources and estimate the capacity of their teams for upcoming sprints, releases, or increments. This helps in balancing workloads and ensuring teams are not overcommitted.
7. **Risk Management:** By visualizing the entire common scope of work and dependencies, you can identify potential risks early in the planning process and take proactive steps to mitigate them.
8. **Resource Allocation:** You can allocate resources to different initiatives and projects based on priority, ensuring that the most critical work gets the necessary resources.

The most common use cases in EG will be cross-team planning, collaboration, and calibration; Dependency tracking and management as well as overview and management of the common endeavors across products.

In this chapter, we will cover the main features and use cases for our EG use. For the rest please refer to the reference materials mentioned at the end of the chapter.

7.11.2 Jira Advanced Roadmaps vs. Aha! Roadmaps

Aha! is our common roadmap platform within EG, used by Product Managers to build high-level roadmaps on an epic level.

Jira Advanced Roadmaps (JAR) can be used by Product Managers and Product Owners to build forecasting and capacity planning, as well as dependency management for a given product on a detailed level. JAR also provides Product Owners with a better overview of, when something can be released, and what is the consequence when something takes a longer time than expected and can thereby better report to the Product Manager of the given status of the forecast and the Product Manager can then prioritize accordingly within Aha!

The workflow is that Product Managers create and prioritize epics in Aha! and agile teams break them down into stories and tasks and make a plan using Jira supported by the Advanced Roadmaps extension whenever necessary.

More about Aha! and Jira integration can be found here: [Jira -> Aha! integration - Next Generation Agile - Confluence EG A/S⁸⁵](#)

7.11.3 Setting and Configuring Jira Advanced Roadmaps plan

7.11.3.1 Creating Jira Plan

To create a Jira plan in JAR go to the "Plans" in the Jira top menu blue bar and from the drop-down menu select: "Create...". From the menu please select "Plan" and press "Create". We don't use "Program" since this level should be done in the AHA! app.

What would you like to create?

Plan
Visualize your roadmaps by automatically or manually scheduling work for your teams, as you see fit.

Program (group of plans)
Track the progress of business initiatives in an aggregated view of work across multiple plans.

Create **Cancel**

Next, provide your Plan name, access permissions (private or public) and specify the issue source. We do recommend using the "private" permissions first that can be extended to the proper permission range later on.

Setting the data (issues) range

The most important thing when setting Jira Advanced Roadmaps is setting the proper range of data (issues) that we want to create a common view for. It should be broad enough to cover all the related issues currently and in the future and narrow enough so we exclude all the issues that are not related to the scope of work we want to manage in JAR.

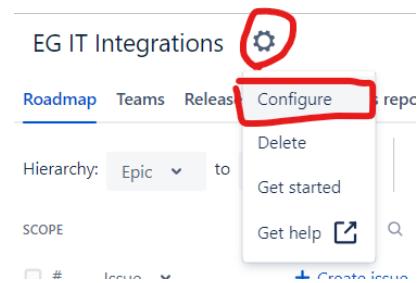
We do recommend you create a specific Jira filter that will cover the required, custom data range. Usually, it is based on a specific epic range, components, teams, or combined multiple factors. You can find more information about the proper filter creation in this chapter: [7.1.10 - Jira JQL hints, useful examples and managing filters - Next Generation Agile - Confluence EG A/S](#) (see page 293)

Please contact the NGA team if you have any issues with the specific, custom filter creation.

7.11.3.2 Configuring Jira Plan

As soon as your Jira plan is created, you can configure it based on your preferences. The primary configuration is affecting the entire Jira Plans scope and can be done in the configuration menu. To access the menu select the cogwheel icon in the top left menu and from the drop-down select: "Configure"

⁸⁵ <https://confluence.eg.dk/display/NG/Jira+-%3E+Aha%21+integration>



In the configuration menu, you can manage among the others:

- **Issue source** - you can adjust or edit the source of the issues visible on Jira plans
- **Permissions** - here you can add specific users, Jira groups or make the Plans public. It can be done on two levels of permission - viewing and editing
- **Exclusion rules** - here you can exclude from your Plans: a) specific issue types (i.e. Sub-tasks) or b) specific issue statuses (i.e. Not done)
- **Adding custom fields** - In Jira Plans you can specify the Jira fields that you would like to view and edit on Jira Plans. Jira Plans allows you to add only default fields that are available in Jira. You can add them by expanding the "Fields" drop-down:

#	Issue	Status	24/Sep	01/Oct	08/Oct	15/Oct
1	EGA-1064 All integrations to Licens...	IN PROGR...	2			
2	EGA-1081 Salesforce up- and dow...	IN PROGR...		2		
	EGA-843 Send subscription Ite...	DONE			1	
	EGA-1134 Remove existing int...	REFINEM...				1

In some cases, you may need to view and edit also custom fields that we use in our Jira (i.e. "Team name" field). To enable them in the Jira Plans drop-down menu you need to select the cogwheel icon in the top left menu and from the drop-down select: "Configure". From the left side menu select "Custom fields" and on the screen select "adding the custom fields" as marked at below screenshot:



You don't have any custom fields in this plan

Tailor this plan to how your team works by [adding the custom fields](#) that you need.

Newly added custom fields will now be available in the "Fields" drop-down menu.

Adding additional text custom fields in Jira Plans

Jira Plans has the possibility to add an additional custom text fields. It can be used in situations when you want to add some comment, text content, or information i.e. as a result of the calibration meeting. This functionality is not available for ordinary Jira Plans users, but you can request the creation of such a field by reaching the NGA DevOps team and specifying the Plans that you are referring to and the additional field name. These fields may be very useful if you want to make specific notes on the tickets in Plans.

The additional custom text fields will be visible only on Jira Plans.

7.11.3.3 Creating and managing views

Jira Advanced Roadmaps allows you to create multiple views that can be useful in multiple contexts which you will use it for. For example, a different view will be used for the recurring calibration meetings and a different one will be more suitable for the planning and issues distribution meeting or dependency review.

You can set, manage, and save distinct views based on the below parameters:

- **Issue hierarchy** - the system allows you to set the range of Jira items from Epics to sub-tasks:

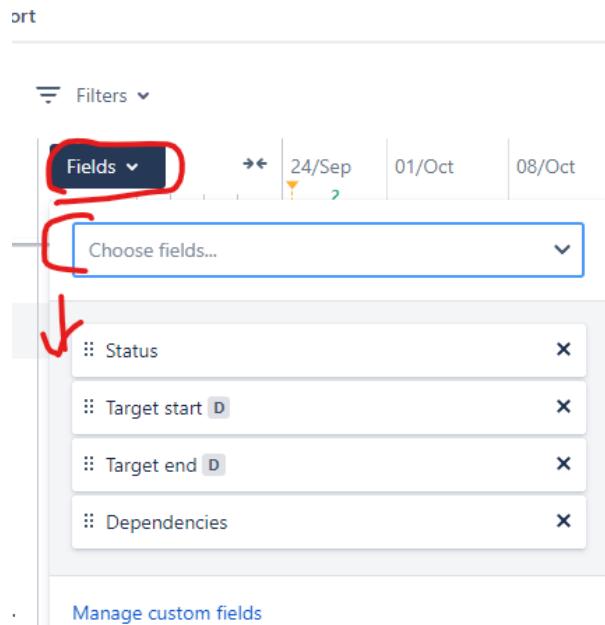
Roadmap Teams Releases Dependencies report

Hierarchy: Epic ▾ to Sub-task ▾

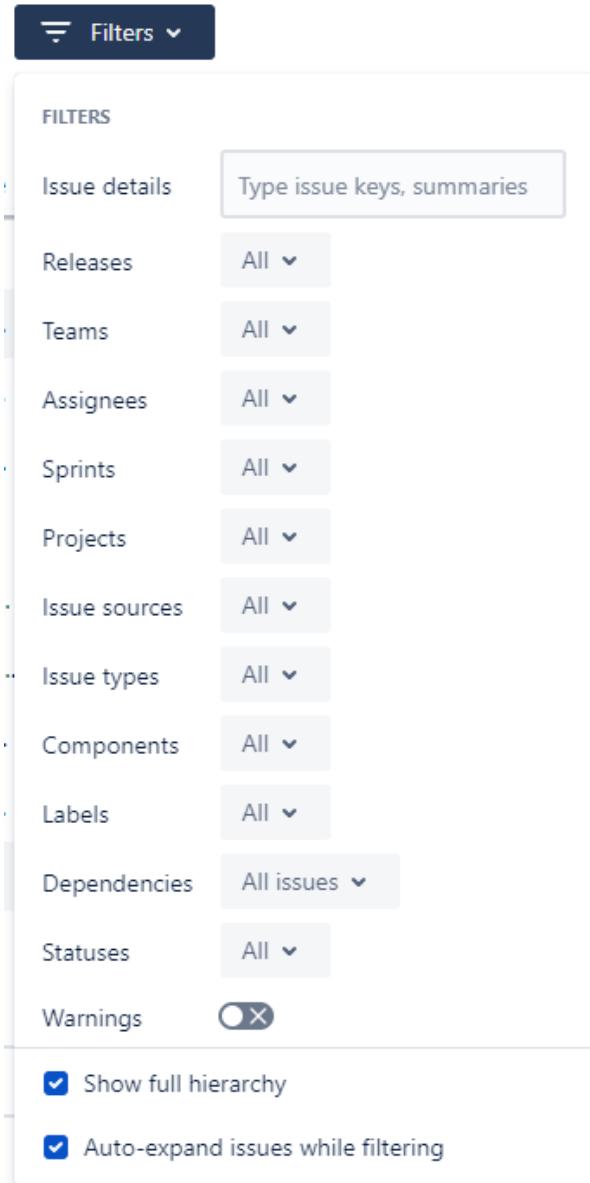
SCOPE

Issue ▾ + Create issue

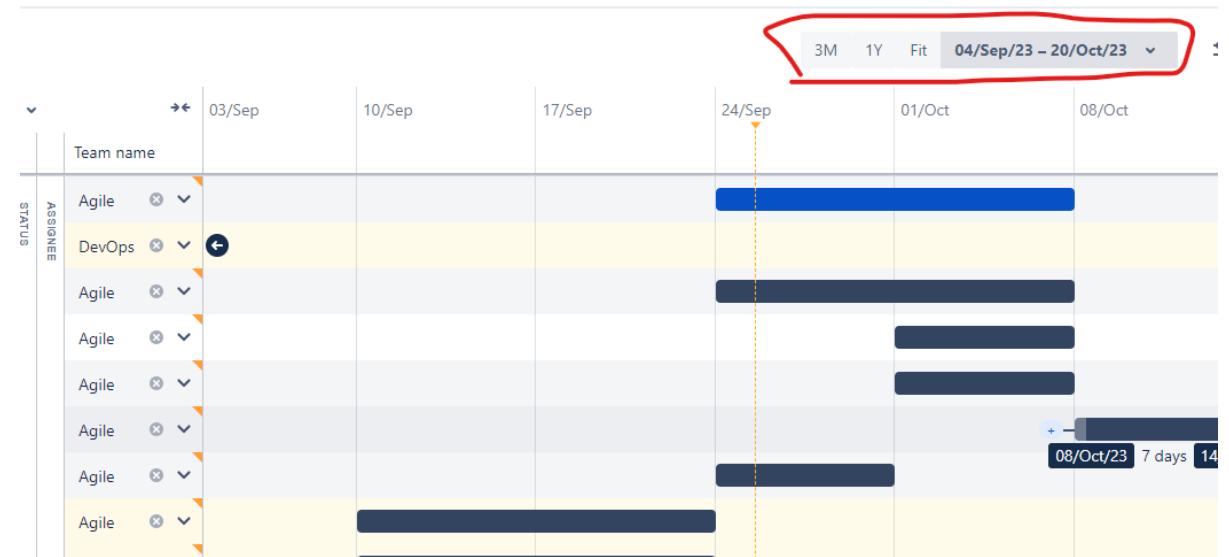
- **Fields range** (including above-described custom fields). You can add, remove or group the fields to reflect your needs:



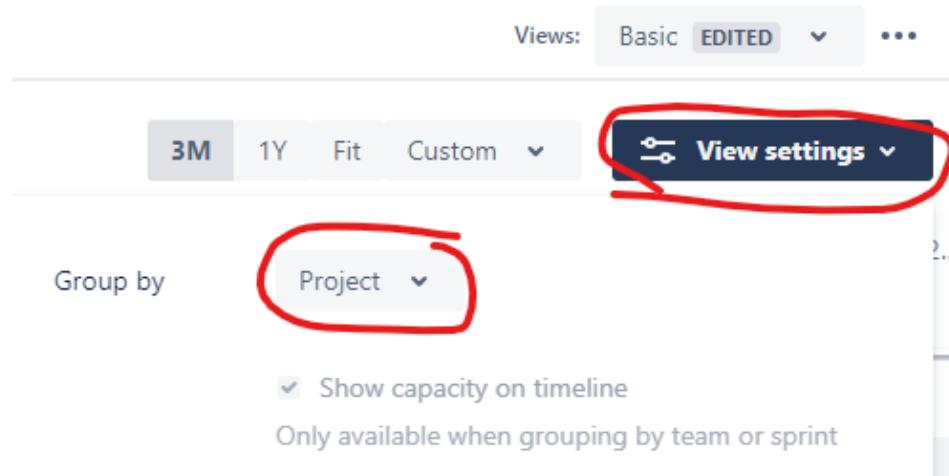
- **Multiple factor filters** - the range of the filters is quite broad. You can combine multiple filters at once. Please look at the below screenshot:



- **Timeline viewing perspective** - Jira Advanced Roadmaps has the possibility to set the planned date range (target start and target end) in a Gantt chart-like style for the issue delivery. It also has the possibility to set a specific time range for this view:



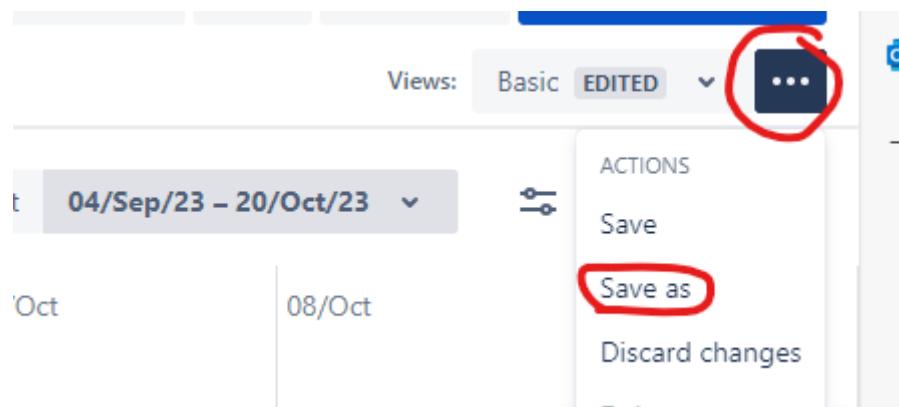
Additionally, you can **group all the viewed data under the swimlanes**. The grouping can be made based on: projects, assignees, components, labels, releases, sprints or teams. For example - if your Plans are gathering the items from multiple Jira projects, grouping them under distinct swimlanes can improve the visibility, discussion and calibration. To access the viewing settings go to the "View settings" right side drop-down and from the expand select: "Group by":



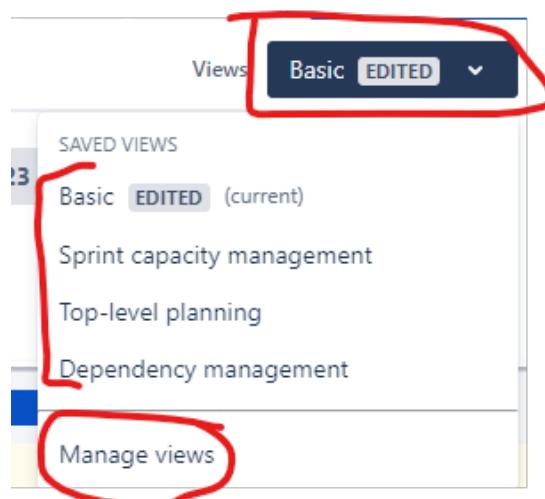
You can use the **issue card colors** to distinguish the items on Jira Advanced Roadmaps based on: status, component, issue type, label, project or team. This can be done under the abovementioned "View settings".

7.11.3.4 Saving and managing saved views

The views that have been properly configured can be saved, so you can access them with a single click to change the viewing context. To save the specific view select three dots and from the drop-down select "Save as". Provide the title for your view. You can make a specific view default for everyone by selecting the respective checkbox. If you want to save the changes to the already created view, select "Save" from the menu:



To switch between the views go to the "Views" panel on the right side of the screen and select the previously saved view. You can also see which view is the default one. If you want to delete or rename a view, select the "Manage views" option in the drop-down:

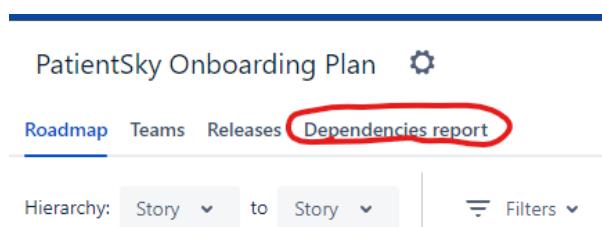


7.11.4 Dependency management

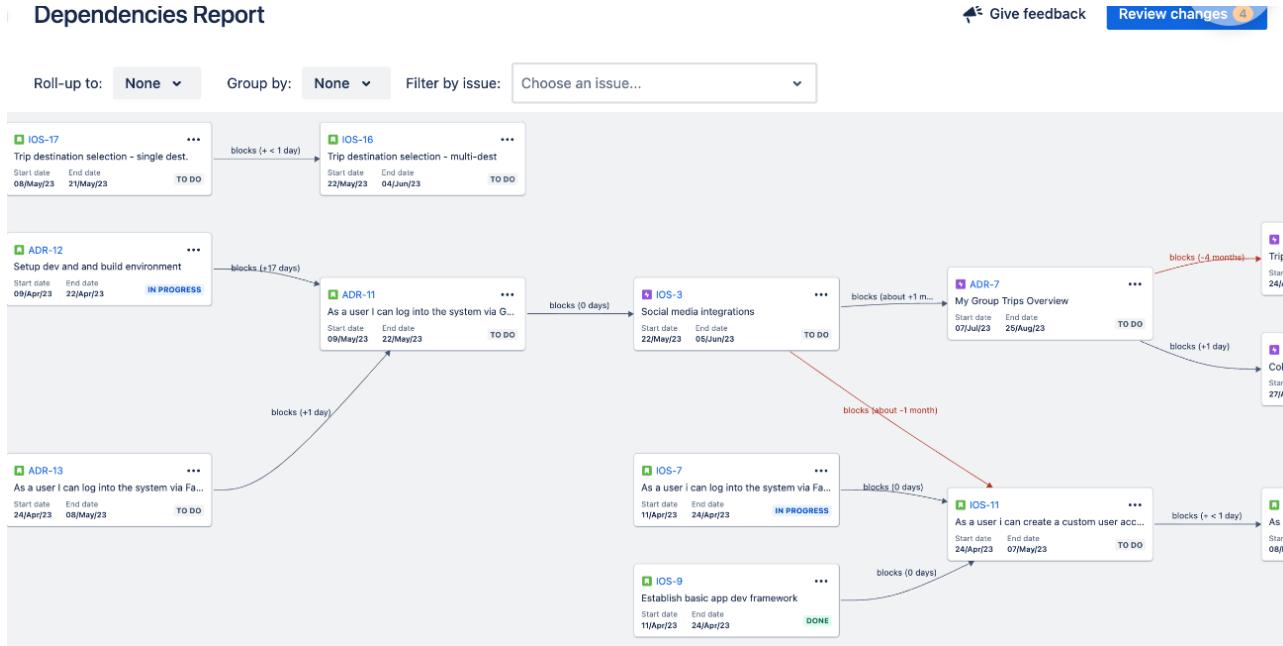
One of the big advantages of Jira Advanced Roadmaps is dependency visualization and management. In EG Jira setup we can review two types of dependencies: "Blocks" and "Relates to".

7.11.4.1 Viewing the dependencies

One of the methods of viewing the dependencies in Jira Advanced Roadmaps is the "Dependencies report" which is available under the dedicated tab:



The Dependency report displays issues from your plan as tiles with arrows to illustrate which issues are dependent on others. If the dependency generates a warning (i.e. in the situation when the blocking item has not yet been finished and the dependant item should already be started), such as misaligned dates, the arrow will turn red. Here is the dependency report/map view:



On the main screen view, the dependencies can be seen as lines linking the issues:



7.11.4.2 Add a dependency

To add a dependency to an issue in your plan:

1. Hover on the schedule bar for the issue to which you want to add a dependency, and select the **+** icon. Selecting the **+** icon on the right will create an outgoing dependency, while the left will create an incoming dependency.
2. Choose the issue to which you want to attach the dependency.
3. Select the **checkmark** to confirm your choice.

After adding a dependency to an issue, adjust the dates by clicking and dragging the ends of the schedule bar. The badges on either end will turn red if there are conflicting dates.

Type

Issue

Status **Assignee** **Lead time**

Blocks

DS-1 Decide which platform to migrate to

TO DO **-6 days**

+ Add dependency

Filter by dependencies of DS-3 Analyze pros and cons of the platform options for migration

Warnings and scheduling of dependencies are based on whether your plan is configured to handle [dependencies sequentially or concurrently](#)⁸⁶.

7.11.4.3 Remove dependencies

Removing dependencies from issues will only dissolve the relationship between the issues. It won't remove the issues from the plan.

To remove dependencies from an issue:

1. Display an issue's dependency details by hovering over the dependency badge.
2. Select the **delete** icon next to the dependency

7.11.5 Managing releases

Jira Advanced Roadmaps allows you to create, manage, and track the progress of the releases. To have this possibility **you must use a fix version feature in Jira**. You need the Advanced Roadmaps **user** permission to save changes in Jira Software. Additionally, you will need **Administer Projects** permission in all projects associated with the cross-project release.

More information about the releases management can be found under this link: [Releases in Advanced Roadmaps | Jira Software Data Center and Server 9.11 | Atlassian Documentation](#)⁸⁷

7.11.6 Reference materials

If you want to dive deep into the specific areas or activities in Jira plans please refer to these resources:

- [Atlassian video tutorials on multiple topics](#)⁸⁸
- [Atlassian Jira Advanced Roadmaps guide](#)⁸⁹

⁸⁶ <https://confluence.atlassian.com/jirasoftwareserver/configure-your-advanced-roadmaps-plan-settings-1044784158.html#ConfigureyourAdvancedRoadmapsplansettings-Dependencies>

⁸⁷ <https://confluence.atlassian.com/jirasoftwareserver/releases-in-advanced-roadmaps-1044784195.html#:~:text=Releases%20in%20Advanced%20Roadmaps%201%20Release%20types%3A%20Project,releases%20...%206%20Remove%20and%20delete%20releases%20>

⁸⁸ <https://www.atlassian.com/software/jira/guides/advanced-roadmaps/tutorials#create-a-plan-in-advanced-roadmaps>

⁸⁹ <https://www.atlassian.com/software/jira/guides/advanced-roadmaps/overview#what-are-advanced-roadmaps>

7.12 7.12 - EG Security default schema for GDPR data access

7.12.1 Introduction

As EG grows and expands its employee base outside of the European Union, it has become even more relevant to ensure that no personal data - in the below referred to as GDPR data - is accessible to those without necessary permissions during the software development process. The following concept presents a solution based on Jira Security schema for distinguishing potentially GDPR issues in Jira and by default permitting access to such issues for employees meeting the mandatory conditions (such as no access to customer data outside an EU location).

Personal data (GDPR data) is any information that relates to an identified or identifiable living individual which collected together can lead to the identification of a specific person. For more information about the definition of personal data, please see the [GDPR Employee Handbook](#)⁹⁰

7.12.2 Solution description

The solution is based on creating a global EG GDPR Security scheme, which will be a part of the EG Default Project Template for creating any future EG project in Jira.

EG GDPR Security scheme Default EG GDPR access Security Scheme	• EG Default Project Template	Security Levels Copy Edit
---	-------------------------------	------------------------------

The Security schema has 2 security levels:

- EU GDPR - for access to issues with GDPR data
- ALL EG - for access to issues without GDPR data (by default all users with Jira access can see issues with this level)

The **EU GDPR security level** - when set - means that only users in the *USR_Atlassian_EG_GDPR_Compliant* group will be able to see and access an issue with that security level.

By default, when creating issues:

- the EU GDPR security level is set if a user is part of the *USR_Atlassian_EG_GDPR_Compliant* group and the issue may potentially contain GDPR sensitive data; the security level can be modified by such user belonging to the *USR_Atlassian_EG_GDPR_Compliant* group at the point of creation of the issue or at a later time
- the ALL EG security level is set if a user is **NOT** part of the *USR_Atlassian_EG_GDPR_Compliant* group and hence by assumption should not contain GDPR sensitive data

⁹⁰<https://eg.controlmanager.net/Docs/LoadPDFReport?docType=2&docID=2748&isImported=true&SPLibraryID=00000000-0000-0000-0000-000000000000&UniqueID=33b13ebd-9ddb-4aa8-b7c8-86e9d95760ae&showVisibleTrackChanges=false&objectID=undefined&objectType=undefined>

Edit Issue Security Levels
USED BY 1 PROJECT

On this page you can create and delete the issue security levels for the "EG GDPR Security scheme" issue security scheme. Each security level can have users/groups assigned to them.

An issue can then be assigned a Security Level. This ensures only users who are assigned to this security level may view the issue.

Once you have set up some Security Levels, be sure to grant the "Set Issue Security" permission to relevant users.

- View all **Issue Security schemes**
- Change default security level to "None"

Security Level	Users / Groups / Project Roles	Actions
ALL EG No access to GDPR sensitive data	• Application access (Any logged in user) (Delete)	Add Default Delete
EU GDPR (Default) Access to GDPR sensitive data	• Group (USR_EG_GDPR_Compliant) (Delete)	Add Edit Delete

Add Security Level

Add a new security level by entering a name and description below.

Name
Description

Add Security Level

The security level is visible on a Jira issue as below visuals.

The security level field on a Jira issue is mandatory, meaning it needs to be set to one of the two mentioned earlier values. If a *None* value is selected, an error message will appear to the user.

This is configured in the EG Default Field Configuration Schema for all field configurations.

EG Default Project Template / EDPT-4
test4

Details

Type:	Story	Status:	TO DO (View Workflow)
Priority:	Low	Resolution:	Unresolved
Labels:	None	Security Level:	EU GDPR (Access to GDPR sensitive data)
GDPR Change Type:	1 - Normal		
GDPR Data	2 - Normal data - INTERNAL		
Classification:			

EG Default Project Template / EDPT-4
test4

Details

Type:	Story	Status:	TO DO (View Workflow)
Priority:	Low	Resolution:	Unresolved
Labels:	None	Security Level:	None
GDPR Change Type:	1 - Normal		
GDPR Data	2 - Normal data - INTERNAL		
Classification:			

Security Level is required.

The possibility to manage the Security Level will be available to Jira project Administrators and users who are part of the *USR_Atlassian_EG_GDPR_Compliant* group. This project role or otherwise defined group is defined in the EG Default Permission scheme for Jira.

Set Issue Security	Project role
Ability to set the level of security on an issue so that only people in that security level can see the issue.	

IMPORTANT INFORATION: Please be aware of the GDPR rules that apply if the employee's primary workplace is outside the EU/EEA:

It is not allowed for off-shore employee (users outside EU) to access or process personal data unless the following is in place:

- a risk assessment regarding transfer of personal data to third country is carried out and approved by Group Legal & Compliance
- approval from the customer is given in advance if the user will get access to customers customer personal data

Example: A developer team sitting in India working for a Business unit where customers do not accept transfer of personal data to third countries. In this case it is important that all personal data are anonymized before sharing the jira issue with the Indian team.

7.12.2.1 Required actions in EG Active Directory configuration

The described solution is based on AD groups, meaning it is independent of the Jira project roles making it easier for new & existing employees to be automatically qualified to a specific group, allowing or not allowing them access to a defined security level.

- a) for all employees who **should** have access to issues with security level **EU GDPR** (meaning they can access potentially GDPR data) a group *USR_Atlassian_EG_GDPR_Compliant* is created in EG AD; these employees should be added to this group upon onboarding to EG
- b) all employees with existing Jira access will be added to the *USR_Atlassian_EG_GDPR_Compliant* by default group upon release of this solution into Jira production environment. Responsible managers will have a possibility to request removal of the existing users who should not be included in the group via a ServiceDesk general request.

7.12.3 Summary

The above described concept assumes:

- a) the list of people with access to GDPR data in EG Jira projects is controlled by belonging to the *USR_Atlassian_EG_GDPR_Compliant* group;
- b) upon onboarding to EG, new employees are either added or not to the above group by the hiring manager, **based on conditions defined by EG Legal & Compliance**
- c) by default all newly created Jira issues will be marked with a EG GDPR security level marking the issues as potentially consisting of GDPR sensitive data unless they are created by users outside of the the *USR_Atlassian_EG_GDPR_Compliant* group

7.13 7.13 - AI Metrics for software development

7.13.1 Introduction

In midst of the rising demand and use of the generative AI capabilities in EG software development, it is crucial to maintain some form of a measure of that impact on our work. Generative AI benefits come with an investment cost, for such tools as CoPilot, Functionize, etc., hence it is necessary to attempt to recognize the benefits and value that these investment introduce into our software development lifecycle. The following is an attempt to standardize and introduce basic metrics for measurement of the change that AI brings to EG teams; this is a concept exceptionally important in the early adoption phase, to recognize the tilt in the results, before it becomes an integral part of every development teams' working process (planning, execution, review). The 4 metrics below focus on measuring the pace and average time of feature delivery by EG agile teams as well as the trend and resolution time for defects in their work.

The data for these metrics will be sourced from Atlassian Jira and processed into a Cognos dashboard. We will choose 3-4 teams to start off with the use of these metrics, then gather the feedback and conduct necessary adjustments before launching this to all agile software development teams in EG.

7.13.2 Metrics overview

Name	Description	Definition
Velocity	Pace of issue delivery/completion	Based on hours or story points, expressed as a sum of those values for all issues (type: Story, Task, Bug) that have been completed (status Done) within a (default: month) time period
Defect Count Delta	Defect creation and resolution trend	Based on issue count of type Bug created vs. issue count of type Bug resolved (status Done or Not Done) with a (default: month) time period
Average Defect Resolution Time	Average time for defects resolution	Based on time between the date of creation of an issue of type Bug (To Do) to the date of its resolution (Done, Not Done); expressed as an average within a (default: month) time period

Name	Description	Definition
Lead Time for User Stories	Average time of user story delivery	Based on time between the date of creation of an issue of type Story (To Do) to the date of its release to production (Done or fixVersion release date if available); expressed as an average within a (default: month) time period

7.13.3 Metrics measurement implementation

Metric name	Jira source data	Cognos presentation
Velocity	<p>a) Sum of story points (<i>field Story Points</i>) b) Sum of hours (<i>field Original Estimate</i>)</p> <p>For issues of type Story, Bug, Task (<i>field Issuetype</i>) in a selected Jira project (<i>field Project</i>) that have been completed within the defined time period</p> <p>Selected team as an option to refine results (<i>field Team name</i>)</p>	<p>Bar graph with timeline defined by the user (by default monthly intervals for the scope of the last 6 months)</p> <p>Toggle or switch presenting SP or Hour values depending on project use (to be selected by user)</p> <p>Selection of Jira project</p> <p>Optional selection of team name within the selected Jira project</p>
Defect Count Delta	<p>Count of issues of type Bug (<i>field Issuetype = Bug</i>) with creation (<i>field createdDate</i>)</p> <p>Count of issues of type Bug (<i>field Issuetype = Bug</i>) with resolution (<i>field resolutionDate</i>)</p> <p>In a selected Jira project (<i>field Project</i>) that have been created or resolved within the defined time period</p> <p>Selected team as an option to refine results (<i>field Team name</i>)</p>	<p>Double line graph outlining the gap between Created - Resolved defects with timeline defined by the user (by default monthly values for the scope of the last 6 months)</p> <p>Selection of Jira project</p> <p>Optional selection of team name within the selected Jira project</p>

Metric name	Jira source data	Cognos presentation
Average Defect Resolution Time	<p>Difference between issue resolution (<i>field resolutionDate</i>) and issue creation (<i>field createdDate</i>) expressed in time</p> <p>For issues of type Bug (<i>field Issuetype</i>) in a selected Jira project (<i>field Project</i>) that have been completed within the defined time period</p> <p>Selected team as an option to refine results (<i>field Team name</i>)</p>	<p>Bar graph with timeline defined by the user (by default monthly intervals for the scope of the last 6 months)</p> <p>Selection of Jira project</p> <p>Optional selection of team name within the selected Jira project</p>
Lead Time for User Stories	<p>a) Difference between issue resolution (<i>field resolutionDate</i>) and issue creation (<i>field createdDate</i>) expressed in time</p> <p>b) Difference between release date of the first release (<i>field fixVersion/s</i>) assigned to the issue and issue creation (<i>field createdDate</i>) expressed in time if the field is not empty</p> <p>For issues of type Story (<i>field Issuetype</i>) in a selected Jira project (<i>field Project</i>) that have been completed within the defined time period</p> <p>Selected team as an option to refine results (<i>field Team name</i>)</p>	<p>Bar graph with timeline defined by the user (by default monthly intervals for the scope of the last 6 months)</p> <p>Toggle or switch presenting date to resolution or date to release values depending on project use (to be selected by user)</p> <p>Selection of Jira project</p> <p>Optional selection of team name within the selected Jira project</p>

7.13.4 Expected measurement results

The forecasted metrics' behavior over a defined period of time after introducing AI supported tools (GitHub CoPilot, GitHub Chat, Functionize) by a project/team.

Metric name	Expected change & duration	Justification of forecast
Velocity	Velocity is expected to gradually increase within 6 months of the team's start of using AI supported development tools. After that time Velocity may drop and stabilize at a slightly lower level, but higher than the initial Velocity.	The gradual increase is due to the team's learning curve on using the introduced tooling efficiently - in some cases a minor drop in Velocity is justified as the learning may consume some of the team's capacity vs. productive output. After reaching peak Velocity, the value will most likely drop insignificantly and stabilize as the AI support will become organically accounted for in the team's estimations of new tasks.
Default Count Delta	Bugs created are expected to remain without change for an approximate period of 3-6 months, after which they should start showing a decreasing trend. Bugs resolved should gradually increase as there is a correlation to the Velocity metric but depends on prioritization in the team's backlog.	It will take some time until the number of created bugs will start to decrease, as the direct result of introducing AI supported tools, which in turn will ensure better quality code that will generate less defects. The increase of resolved # of bugs should be observable sooner similarly to the rising Velocity, hence closing the gap between Bugs created vs. Bugs resolved to a positive bias.
Average Defect Resolution Time	ADRT is expected to gradually increase within 3-6 months of team's start of using AI supported development tools.	The gradual increase is due to the team's learning curve on using the introduced tooling efficiently. Results on resolution time of issues should be observable the soonest of all metrics, as this is where AI tools' support should strive in guiding towards the solution of a problem quicker, which in turn will allow resolving more defects in a shorter period of time.
Lead Time for User Stories	Lead Time is expected to gradually increase within 6 months of the team's start of using AI supported development tools.	The gradual increase is due to the team's learning curve on using the introduced tooling efficiently. Compared to ADRT, the results may be observable a bit later, due to the higher comparable complexity of the tasks. However, the reached improvement on the Lead Time will similarly allow more frequent User Story delivery (depending on team's release policy).

8 8 - Appendix

- [8.1 - Glossary \(see page 414\)](#)
- [8.2 - FAQ \(see page 420\)](#)
- [8.3 - Examples \(see page 427\)](#)
 - [8.3.1 Retrospective \(see page 427\)](#)
 - [8.3.2 Product and Sprint Goal \(see page 433\)](#)
- [8.4 - EG Agile Certification program \(see page 437\)](#)

8.1 8.1 - Glossary

8.1.1 A

Agile Coach: a person responsible for fostering the agile mindset, spreading agile values and principles throughout the organization's working environment with a specific focus on continuous improvement. The Agile Coach works to support building teams using Scrum as one of the frameworks, as well as to assess and drive the teams' maturities through mentoring and supporting their self-organization. [3.5 - The role of an Agile Coach \(see page 88\)](#)

Agile Manifesto: a brief document built on 4 values and 12 principles for agile software development, published in February 2001. [2.2 - The Agile Manifesto and change of mindset \(see page 39\)](#)

Assignee: a person who is currently assigned to the Sprint Backlog Item or Product Backlog Item.

8.1.2 B

Burn-down Chart: a chart that shows the amount of work that is thought to remain in a backlog. Time is shown on the horizontal axis and work remaining on the vertical axis. As time progresses and items are drawn from the backlog and completed, a plotline showing work remaining may be expected to fall. The amount of work may be assessed in any of several ways such as user story points or task hours. Work remaining in Sprint Backlogs and Product Backlogs may be communicated by means of a burn-down chart.

Burn-up Chart: a chart that shows the amount of work that has been completed. Time is shown on the horizontal axis and work completed on the vertical axis. As time progresses and items are drawn from the backlog and completed, a plotline showing the work done may be expected to rise. The amount of work may be assessed in any of several ways such as user story points or task hours. The amount of work considered to be in-scope may also be plotted as a line; the burn-up can be expected to approach this line as work is completed.

Bug: or defect is an item in the Product or Sprint Backlog that represent an improper functioning, malfunctioning or lack of functioning of a feature expressed within a previously implemented backlog item.

Business Project: in EG is a grouping of multiple Product Backlog items (usually Epics, but may also include User Stories) across 1 or more products. The grouping is done with a Jira issue type business project not to confuse with the naming convention of a Jira project which is a technical representation of an EG product on the Jira platform.

8.1.3 C

Capacity: the amount of work that can be handled by the team in the sprint. That amount of work is usually expressed as a ratio of hours available in the sprint-defined time period and hours that Developers plan to dedicate for executing sprint-related tasks.

Common Language: a set of rules that define how we work and deliver software products throughout the entire EG company. A unified, common language is helpful to build a coherent understanding and working culture across all Business Units, which will lead to a closer integration within EG. [2.3 - The common language \(see page 40\)](#)

CTO: EG's Chief Technology Office who is lead by our Chief Technology Officer [@Allan Bech](#). It's our CTO that own EG's Next Generation Agile (NGA) program.

Continuous integration: is the practice of merging all developers' working copies to a shared master branch as frequently as possible (several times a day).

8.1.4 D

Daily Scrum: daily time-boxed event of 15 minutes for the Developers to inform what has been done in the previous day and to re-plan the next day of development work during a Sprint. Updates are reflected in the Sprint Backlog. Each participant is usually answering the Three Questions.

Definition of Done (DoD): a shared understanding of expectations that the Increment must live up to in order to consider the item Done. Definition of Done is agreed between the Product Owner and the Developers.

Definition of Ready (DoR): the minimum requirements list for any Product Backlog item to be considered Ready in order to take it into the Sprint and start the development process.

Demo: part of the Sprint Review meeting where the Developers are presenting a working software as a potentially releasable Product Increment.

Developers: the role within a Scrum Team accountable for managing, organizing and doing all development work required to create a releasable Increment of product every Sprint.

DevOps Engineer: a person who has combined knowledge of software development (Dev) and information-technology operations (Ops). It's a person who aims to shorten the systems development life cycle and provide continuous delivery, increasing the software quality at the same time.

Done: the status of the particular item which has been passed through the Definition of Done and has been considered by the Product Owner delivered as expected.

8.1.5 E

Effective work time: the time (number of hours and minutes) that are considered to be fully dedicated to the assigned work. Usually, it is between 6 and 7 hours.

Empiricism: process control type in which only the past is accepted as certain and in which decisions are based on observation, experience, and experimentation. Empiricism has three pillars: transparency, inspection, and adaptation.

Epic: a large body of work that is eventually broken down into smaller User Stories when adding the details. Epics are often used as placeholders for new ideas that have not yet been decomposed into User Stories. Epics are also being used to gather together User Stories to better organize the Product Backlog.

Estimation: the process of agreeing on a size measurement for the Product Backlog Items, done by the Developers.

8.1.6 F

Fibonacci Sequence the sequence of numbers where the next number is derived by adding together the previous two (1,2,3,5,8,13,20...). The sequence has the quality of each interval getting larger as the numbers increase. The sequence is often used for Story Points because estimates are always less accurate when dealing with larger items.

Forecast (of functionality): the selection of items from the Product Backlog that Developers deems feasible for implementation in a Sprint.

8.1.7 I

Impediment: any external or internal (within the team) factor that prevents the Developers from creating the Product Increment and meeting their Sprint Goal.

Increment: a piece of working software that adds to previously created Increments, where the sum of all Increments -as a whole - form a product.

I-N-V-E-S-T mnemonic: a set of guidelines that helps guarantee the quality of user stories or other backlog items. The letters in INVEST stand for: Independent, Negotiable, Valuable, Estimable, Small, Testable.

8.1.8 M

Minimum Viable Product (MVP): the smallest, most fundamental version of the Product, which has business value and can be made available to the customer.

MoSCoW: it is a technique used to prioritize the Product Backlog. The term *MoSCoW* itself is an acronym derived from the first letter of each of four prioritization categories (*Must have*, *Should have*, *Could have*, and *Won't have*).

8.1.9 N

NGA: The Next Generation Agile program, also called NGA, is launched with the vision to empower our organization to create and improve best-in-class software solutions and deliver at a faster pace to our customers.

8.1.10 P

Pair Programming: occurs when 2 (a pair) of Developers on the team work together on 1 workstation (potentially using a second one for reference, sanity checking or reviewing) to deliver a single decomposed piece of work.

Planning Poker: one of the techniques used to evaluate the approximate effort of a Product Backlog item using story points. [5.2 - Refining and estimating an initial Product Backlog - Next Generation Agile - Confluence EG A/S \(see page 152\)](#)

Pull-request: Pull-request (also known as merge-request) is a way to make sure all code is reviewed before it is merged and potential shipped to production. A pull-request is to request that code-changes are merged into another branch. The Pull-request can be accepted and the code is merged or rejected with no merge.

Pre-planning: When working with multiple Scrum Teams and having cross-team collaboration, Pre-Planning is part of that process. Here items for next refinement are identified. [6.2 - Cross-team collaboration in a single product \(see page 219\)](#)

Product: In general, a product in EG is usually a system (simple or more complex, composed of components) or a piece of software which is being continuously developed, improved and delivered to multiple customers.

Product Backlog: an ordered list of the work to be done in order to create, maintain and sustain a product. Managed by the Product Owner.

Product Backlog Item: common name of the items that can be created in the Product Backlog, which contains: epics, user stories, tasks, bugs, and sub-tasks

Product Backlog Refinement: the activity in a Sprint through which the Product Owner and the Developers add granularity and details to the Product Backlog and checking if the particular items are Ready to be taken into the Sprint. The refinement process should not take more than 10% of the capacity of the Developers.

Product Increment: the sum of all the Product Backlog Items completed during a Sprint and the value of the increments of all previous Sprints.

Product Manager: A role specified that collaborates with the Product Owner, focusing on the Product Vision and removing higher-level impediments. [3.3 - The role of a Product Owner and Product Manager \(see page 77\)](#)

Product Owner: a person who holds the vision for the product and is responsible for maintaining, prioritizing and updating and managing the value of the Product Backlog. In Scrum, the Product Owner has final authority representing the customer's interest in backlog prioritization and requirements questions.

8.1.11 R

RACI model: a straightforward tool used for identifying roles and responsibilities and avoiding confusion over those roles and responsibilities during a project. The acronym RACI stands for: Responsible, Accountable, Consulted and Informed.

Ready: a shared understanding by the Product Owner and the Developers regarding the preferred level of description of Product Backlog items introduced at Sprint Planning.

Refinement: see Product Backlog Refinement.

Retrospective: see Sprint Retrospective.

Retrospective improvement: improvement selected and agreed by the Scrum Team which are the outcome of the previous Sprint Retrospective meeting that will be addressed in the next sprint to make the Scrum Team work better and more productive.

8.1.12 S

Sanity check: part of the Sprint Planning meeting where the Developers are validating if the sum of planned Sprint Backlog scope delivery time corresponds to the team's capacity

Scrum: A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.

Scrum Guide: the definition of Scrum, written and provided by Ken Schwaber and Jeff Sutherland, available under this link: <https://www.scrum.org/resources/scrum-guide?>

gclid=CjwKCAiA1rPyBRAEiwA1Ul8L3KIRJd-

NHlmyuU0SXQJkxrN_EReHNo4cj74pse8Q8eRzfByhwUyRoCXGcQAvD_BwE. This definition consists of Scrum's roles, events, artifacts, and the rules that bind them together.

Scrum Master: the role within a Scrum Team accountable for guiding, coaching, teaching and assisting a Scrum Team and its environments in a proper understanding and use of Scrum.

Scrum of Scrums (SoS): If cross-team collaboration is needed, An event that follows the topics from Daily-Scrum principles is held with representation from the teams. Duration is 30 minutes and frequency is once a week. [6.2 - Cross-team collaboration in a single product \(see page 219\)](#)

Scrum Pillars: Scrum is founded on empirical process control theory, or empiricism. Empiricism asserts that knowledge comes from experience and making decisions based on what is known. Scrum employs an iterative, incremental approach to optimize predictability and control risk. Three pillars uphold every implementation of empirical process control: transparency, inspection, and adaptation.

Scrum Team: a self-organizing team consisting of a Product Owner, Developers and Scrum Master.

Scrum Values: a set of fundamental values and qualities underpinning the Scrum framework; commitment, focus, openness, respect and courage.

Self-organization: the management principle that teams autonomously organize their work. Self-organization happens within boundaries and against given goals. Teams choose how best to accomplish their work, rather than being directed by others outside the team.

Slack time: is all the time, which is intentionally or unintentionally used for various types of self-organization, growth, and improvements within the Developers' work (examples: knowledge transfer, cross-competencies training, research and experimentation, reduction of technical debt, improvement of processes and tools within the team).

Spike: a special type of work that is used to gain the knowledge necessary to reduce the risk of a technical approach, better understand a requirement, or increase the reliability of a story estimate.

Sprint: time-boxed event of one month or less, that serves as a container for the other Scrum events and activities. Sprints are done consecutively, without intermediate gaps.

Sprint Backlog: an overview of the development work to realize a Sprint's goal, typically a forecast of functionality and the work needed to deliver that functionality. Managed by the Developers.

Sprint Backlog Item: a Product Backlog Item that has been assigned to the current Sprint at the Sprint Planning meeting or after agreeing after it. This can be user story, task, bug or sub-task.

Sprint Goal: a short expression of the purpose of a Sprint, often a business problem that is addressed. Functionality might be adjusted during the Sprint in order to achieve the Sprint Goal.

Sprint Planning: time-boxed event of 8 hours, or less, to start a Sprint. It serves for the Scrum Team to inspect the work from the Product Backlog that's most valuable to be done next and design that work into Sprint backlog.

Sprint Priorities: reflected in the Sprint Backlog order of the all Sprint Backlog items - most important items at the top, less important at the bottom.

Sprint Retrospective: time-boxed event of 3 hours, or less at the end of a Sprint. It serves for the Scrum Team to inspect the past Sprint and all the improvement areas and plan for improvements to be enacted during the next Sprint.

Sprint Review: time-boxed event of 4 hours, or less, to conclude the development work of a Sprint. It serves for the Scrum Team and the stakeholders to inspect the Increment of the product resulting from the Sprint, assess the impact of the work performed on overall progress and update the Product Backlog in order to maximize the value of the next period.

Stakeholder: a person external to the Scrum Team with a specific interest in and knowledge of a product that is required for incremental discovery. Represented by the Product Owner and actively engaged with the Scrum Team at Sprint Review.

Story point: a relative unit of measurement applied to size Product Backlog Items by grouping them into relative size ranges rather than by absolute units (e.g. – hours) i.e. Fibonacci Sequence T-shirt sizes, or other ways of assigning Story Points.

Swarming: occurs when multiple Scrum Team members work on only one Sprint Backlog item at a time during the Sprint. This means that the work of an item can be decomposed in such a way, to make it possible for multiple team members to contribute to the Sprint Backlog Items by realizing one or more of its sub-tasks.

Sub-task: particular small pieces of work that are needed to be done, to complete the User story, Task or Bug. All Sub-tasks are entities that belong to their parent, under which they were created.

8.1.13 T

Team working agreement: a set of rules created and agreed by the entire team related among others to: meetings, team collaboration, coding standards and reviews, working with the backlog, planning and all other aspects that the team would like to set up the rules for.

Team identity: building and maintaining a common team members identification as a Team which helps to bring the team together to focus on a common goal. This may be a team name, team logo and team slogan.

Team page: a common space where the particular Scrum teams can store, track and update all the team related content: information, agreements, rules, activities, notes, reports, etc.

Technical Debt: the typically unpredictable overhead of maintaining the product, often caused by less than ideal design decisions, contributing to the total cost of ownership. May exist unintentionally in the Increment or introduced purposefully to realize value earlier.

Three questions of the daily scrum: What did I do yesterday that helped the Developers meet the Sprint Goal? What will I do today to help the Developers meet the Sprint Goal? Do I see any impediment that prevents me or the Developers from meeting the Sprint Goal? Those questions are being answered during the Daily Scrum meeting yet are not mandatory.

Timeboxing: setting a duration for every meeting or activity and having them last exactly as they have been set.

8.1.14 U

User story: short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template: **As a < type of user >, I want < some goal > so that < some reason >**. The user story represents the business requirement. User stories are the main components of Product Backlog.

8.1.15 V

Values: commitment, courage, focus, openness, and respect embodied and lived by the Scrum Team.

Velocity: indication of the amount of Product Backlog turned into an Increment of product during a Sprint by a Scrum Team, tracked by the Developers for use within the Scrum Team.

Velocity Average: an average out of a series of consecutive Sprints of the Scrum Team, usually limited to 6-12 last Sprint.

8.2 8.2 - FAQ

This section is a continuously updated listing of frequently asked questions, which our team has come across throughout various stages of the NGA program and also those which we anticipate being raised in the future. The FAQ is categorized in several areas to bring focus to the area of interest of the user. Any recommendations or suggestions as to the content of this section are welcome and we encourage everyone to contribute.

NGA General	
Can I still contact the NGA for support once our Scrum Team has been offboarded from the program?	Yes, of course! While the NGA team, and especially your assigned agile coach and devops engineer are most heavily involved with your team during and after the onboarding process, we remain available to you for support even after your team has been offboarded. Please feel free to reach out to us via all available communication channels with any issues that you may have encountered either on the tooling or process side, and we will try to assist you as soon as possible. Sometimes, if the issue is very complicated or time-consuming, we may require to create a ticket related to the resolution of your issue but we will prioritize it accordingly and handle it in our team at the nearest possible moment.
How to contact the NGA team with issues related to the NGA onboarding?	If an issue related to the NGA program is identified, the first thing to do is to check the available materials (such as the EG Agile Playbook, manuals, and training) for a possible resolution or answer to the issue/question. If the answer or resolution was not found in the indicated materials, please reach out to your agile coach or devops engineer of contact - there are always 2 contact persons (1 agile coach and 1 devops engineer) indicated during the NGA kick-off, who are responsible for your business unit; you can reach them via phone, e-mail or ms teams at your convenience. Some issues will be resolved immediately, but others may require more time to properly investigate and propose a solution. In such a case, the person of contact will ask you to generate a ticket in the ONB (Onboarding) Jira project with the proper BU as the epic and "Onboarding issues" as the component. If you do not have access to this project, please ask your manager or someone from the BU change team to do this for you. The ticket will be prioritized and addressed by the NGA team accordingly.

What is the NGA onboarding process and how can I better prepare for what's to come?	<p>The NGA onboarding process consists of several stages, which support the preparation, onboarding and support stages of the process. Initially, a change team is designated in the business unit, which will be the main contact group throughout the preparation until the kick-off. The best you can do is support the BU and NGA team with building the most suited change team, which will best represent the business unit. The first meeting together with the NGA team is during an introductory workshop with the change team, where various aspects of the BU's operation (such as teams, products, processes, tools, etc.) are discussed, for the NGA to better understand the BU and its needs. If you are part of the change team, or would like to give your valuable input to the meeting - the best you can do is to express yourself openly and completely, exposing how product development is done and what are the challenges that you are facing. After the introductory workshop is the one of the most vital parts of the process, where preparatory work is done by the NGA team together with the BU change team, in order to adjust their processes and tools to be compliant with the target NGA setup. During that time, it is important for the change team to be heavily involved in the discussion and collaboration with the NGA team to make all aspects of the changes fit the BUs needs. All input is vital to best prepare for the kick-off; please follow the NGA team's instructions which will guide you throughout this journey. The NGA kick-off is usually a 2-day workshop where all of the (new) Scrum Teams in the BU are onboarded onto the program and start their first sprint in the new way of working. Please be open-minded and participate actively during the prepared workshops to get the most value out of the meetings. Remember that your maturity both on the process and tooling side will grow as you proceed also after the kick-off itself, and the dedicated agile coach and devops engineer will help you get the most out of it for some period (depending on the team, usually 3-6 months) after the kick-off. Therefore feel encouraged to take advantage of the opportunity, grasp the hints and recommendations from your agile coach / devops engineer, don't be afraid to ask questions and ask for help.</p>

Atlassian Jira / Confluence

Why is the "Done" status irreversible in the EG NGA default workflow?	<p>"Done" is a terminal status and hence an irreversible one for several reasons. One of them is to enforce thorough verification of user stories before having them completed, while another is to avoid situations where user story scope is negotiated at a final stage and pinged-ponged back and forth (feature creeping). Final approval of a backlog item has high significance, confirming that such item is delivered as expected (all acceptance criteria are met) by the Product Owner and complies with the EG Definition of Done. We understand that occasionally, due to a human error, this status may be activated by mistake; on such rare occasions, the team may clone the ticket to continue working with proper representation of its' status while maintaining the original issue linked to the clone.</p>
---	--

Why is there no "Test in Progress" or "In Testing" status in the EG NGA default workflow?	The NGA workflow is simple, reflecting the simplicity of Scrum. There is only 1 status, which is responsible for marking all of the work that needs to be done for an item to be completed by the Scrum Team, and that is "In Progress". Testing is just one of the components which composes the completion of a sprint backlog item, and similarly to such activities as documentation, mock-ups, designs, schemas, etc. may be needed in various aspects depending on the particular backlog item. The workflow does not reflect all of the micro-stages necessary for the delivery of a backlog item, because this decision is left to the Developers, which decides what actions are needed to deliver the expected result while being compliant with the EG Definition of Done. All identified "sub-tasks" should be created by the team for each backlog item during the Sprint Planning at the latest and planned accordingly. Test-related activities should be one of these sub-tasks.
How much are we allowed to configure in Jira to fit the team's needs?	The Jira projects created for all of the EG Products in the onboarded business units, represent a common NGA Jira project template, which is part of the NGA Common Language in order to bring unification on the tooling and processes part across EG. That is the reason, why core configurations like the workflow, issue types and custom fields in Jira restricted and cannot be modified by any Scrum Team. However, at NGA we understand that Scrum Teams can differ in their approach and habits which makes some visual and functional aspects of using Jira possible to be configured according to the team's preference. These elements include: order/visibility of standard fields on various Jira issue types, custom dashboards/reports, backlog or active sprint boards view in Jira (especially displaying of additional data on the issue tiles and records). Some of these configurations can be done by a person on the team, however, if support is needed feel free to reach out to your assigned agile coach or devops engineer for help.
Scrum Events & Processes	
If all of the work planned in a Sprint was not completed, can the Sprint be extended?	No, the Sprint event in EG Scrum is a fixed 2-week timebox that is completed after the timebox expires, irrelevant of the result and completion rate. The planned Scrum events in-between the Sprints constitute a crucial part of the process, each having a defined purpose that supports the 3 pillars of Scrum: Transparency, Inspection, and Adaptation. After a completed Sprint (2-week timebox) the Scrum team should evaluate the result of the finished Sprint and re-plan for the next Sprint to come taking into account all items in the Product Backlog, including those not completed in the previous iteration.

How much time should we dedicate to backlog refinement?	<p>The purpose of backlog refinement is to prepare product backlog items in enough detail for them to be considered by the Developers as "Ready" for the upcoming sprints. While the NGA default recommendation is set to 90 minutes per week, in joint Developers' sessions, the backlog refinement is not limited only to those meetings. The process of backlog refinement includes all activities, whether it's in group sessions, individual or pair work, that concentrate on detailing, clarification, estimation and any preparatory work regarding product backlog items, which are anticipated for upcoming sprints. According to the Scrum Guide, backlog refinement should take even up to 10% of the team's capacity each sprint, however it is good practice dedicate as much time as needed in order to have enough product backlog items "Ready" that will allow conducting a successful Sprint Planning. Once the Scrum Team is stable on achieving that goal on a sprint-to-sprint basis, the next step is to continue refinement in order to have some "Ready" backlog items also available for 2 and 3 sprints forward. Once the team has a candidate backlog of "Ready" items sufficient for 3 sprints ahead, the amount of time spend on refinements can be minimized.</p>
How do we go from estimating in hours to forecasting in Story Points?	<p>While for the time being, the estimation techniques vary across EG business units, the NGA long term goal is to focus on a rather relative form of estimation, which will support our agile initiative and a more reliable way of forecasting work. If the decision made with the business unit is to start with relative estimation, using methods such as Story Points then that will be incorporated into the teams' NGA kick-offs to introduce the concept and guide the teams throughout after it. Some BUs or Scrum Teams may needs some time to become adjusted to the new way of working, hence the introduction of Story Points into the process may be delayed accordingly. The readiness and need for going from estimating in hours to Story Points should be consulted and evaluated together with your agile coach. He will provide the necessary trainings, materials and guide you through the process to become an even more mature Scrum Team, with the introduction of relative work forecasting techniques. Switching the technique is just a part of the process, as it requires a general change of mindset and the approach to how work is forecasted and accounted for; also to how using Story Points, the team can benefit from maintaining their velocity and how to use that velocity for a variety of purposes. If you'd like to learn more about relative estimation and Story Points, look into the EG Agile Playbook and reach out to your agile coach.</p>
Scrum Roles / Scrum Team	

Can the Scrum Master role be shared with the Product Owner role by one member of the Scrum Team?	<p>No, a single person who is part of a Scrum Team, may not possess both the Scrum Master and Product Owner roles at the same time. In EG Scrum, the Scrum Master is not a job function, but a role, therefore usually one member of the Scrum Team is granted this privilege and responsibility. Having a team that consists of a Product Owner and various specialists, developers, testers, designers, etc. it must not be the Product Owner, but anyone else from the team who is appointed the Scrum Master role. This person of course needs to be approved by the rest of the team in this role, as it requires trust from the other team members, going forward and working together in this setup. The Product Owner is not suitable to exercise the Scrum Master role as there may exist quite a role conflict between the two, and it would be difficult to remain impartial at challenging moments.</p>
Who among Developers is responsible for testing and documentation?	<p>Developers on the team should be composed of people, who altogether possess all necessary skills and competences to deliver a potentially releasable increment of a Product. The key here is Product and what it consists of; assuming that all software should be properly tested, there should be people on the team with knowledge on how to test software. If the Product requires some special kind of testing and/or the Product requires documentation of different types, then Developers group should be composed of specialists who can provide those elements as part of the end-Product. The responsibility for the outcome rests on the entire team, therefore it is not relevant to have dedicated people to deliver each piece of the Product. Developers should be able to self-organize and agree on the approach to delivering the expected Product with all its requirements. Therefore, similarly to the software being delivered, the testing and documentation need to be done by the entire team as agreed by the entire team: this can mean that all members of the team create/conduct test or all members write documentation, even though their primary focus is coding software. Having dedicated testers or documentation writers in the team is of course allowed, however, the entire team must feel responsible to help each other out to deliver the expected Product when other members of the team are in need of it.</p>

<p>What to do if a Scrum Master or Product Owner is absent during the Scrum Events or for a longer period of time?</p>	<p>Let's remember that the Scrum Master and Product Owner are also human 😊 so it can happen, as for us all, that they need to take a vacation or are out sick. It is often said that a good Scrum Master is recognized not by what he/she does, but how his/her team operates when their Scrum Master is away. Please remember that the Scrum Master is a role in the team, who's big part is to educate and support the team in proper understanding of Scrum - but the entire ownership and responsibility for the following the process and for the end result lies on the entire team. While this may not be valid for a "new" team (in which case, please ask your agile coach for support), a team that has been working for several sprints in the setup, should have sufficient knowledge about the process and should be able to continue the business as usual. The activities and responsibilities, such as the facilitation of Scrum Events, should fall onto the other team member(s) during the absence of their appointed Scrum Master. It is also possible to have a Scrum Master from another team (but same EG Product) help out, but that needs to be a conscious decision taking into account the capacity of that person with respect to his/her first Scrum Team.</p> <p>Regarding the Product Owner role, the situation is a bit different. Since the Product Backlog should not remain without an active Product Owner at any given time, that the Developers group is operational. Should a Product Owner be planning a longer vacation, it must be agreed with the entire Scrum Team who takes over the responsibility and duties of the Product Owner. This may vary from team to team, and can differ case by case; some possibilities are: delegating the role to another Product Owner (who works with a different team in scope of the same product) or to the Product Manager who should have the possibility to temporarily replace the Product Owner. If that is not possible, it could be possible to appoint one of the collaborating consultants who know the product or ultimately (if all else fails) one of the experienced members of the DEV team. Keeping in mind, that even during the absence, the substitute Product Owner should have the empowerment to make binding decisions regarding the product. This decision needs to be made clearly, taking into account all the mentioned needs and after consultation with the DEV team.</p>
Scrum Artifacts	
<p>Can we have our own custom "Definition of Done" and "Definition of Ready" for the Scrum Team?</p>	<p>The "Definition of Done" and "Definition of Ready" are a vital part of the EG Scrum processes which are being introduced by the NGA program. In order to maintain best practices and a coherent vision of what it means for a backlog item to be "Ready" or "Done", we have prepared an EG baseline definition of "Done" and "Ready" to be used across all BUs and Scrum Teams in EG. These definitions can be found in the EG Agile Playbook.</p> <p>While the baseline definitions should not be altered by the BU or the Scrum Teams, they can be expanded upon. This means that the BU or Scrum Team can enrich the DoD or DoR with additional statements that make them more qualitative and error-proof with respect to their product-specific requirements. It is important to remember that a DoD or DoR for multiple teams working on a single product should be the same across those teams, meaning there should not be various DoD or DoR for a single EG product.</p>

How often should we look at the burndown chart and how does it help?	<p>The burndown chart shows the team's progress towards the sprint goal throughout the sprint, considering a reference, stable burndown of "work" compared to the actual burndown of "work" done by the Scrum Team. Looking at the sprint burndown chart (such reports are available in Jira for the entire team) helps the Scrum Team understand where they are as compared to where they should be during the 2-week cadence. This understanding in turn allows the team to discuss if any adjustments to their plan are needed in order to meet the sprint goal and to complete the forecasted work. It is sufficient to have a look at the sprint burndown chart every other day and that can easily be incorporated for example during the Daily Scrum; however, along with the team's maturing, the need for frequent inspection of their burndown decreases, as they become more proficient.</p>
Can we modify the Sprint Backlog?	<p>The Sprint Backlog is a list of work, which is forecasted to be completed by the Scrum Team in the defined Sprint cadence. The issues which are in that list reflect the most prioritized and vital needs of the Product Owner to increase the value of the Product, and should be revised with the entire team, as it is the Developers who decide on the commitment to the above-mentioned forecast. Knowing that it is important to understand that Scrum embraces and acknowledges change, which is a part of all of our daily lives. It of course can happen that the needs of the Product change urgently and suddenly - impossible to foresee. In such a case the most important is to have dialog within the Scrum Team - this can happen ad hoc, or during the Daily Scrum if possible. In that dialog, the whole team needs to respond to the Product (Owner) need and answer the question: "How can we make this happen?" without disturbing the Sprint Goal and our standard processes. Usually, such a requirement implies the need to perform some rotation in the Sprint Backlog priorities; sometimes this will require descoping several items or part of an item (further breakdown) from the Sprint Backlog, to make room for the new request. While that is acceptable and respected, we should always plan the Sprint with the best knowledge available, to complete all of the forecasted work within the fixed timebox; but of course, we need to react to change and replan the Sprint with respect to the new learnings.</p> <p>Another case is when the initially planned team capacity drops significantly, due to unplanned events such as sickness or accident. The natural next step is also to start a dialog within the team: "Can we accommodate for the loss of capacity? If yes, how much of it? How can we do it and what should be descoped if needed (should the Sprint Backlog order be revised)" - those are the questions that the team needs to answer. Once that is accomplished, you should have a clear vision of the new plan for the sprint, and at the same time the Product Owner will be aware which items from the backlog will be affected by this capacity loss - that will allow him/her to adjust his/her potential obligations to the stakeholders. Issues that are deemed not possible to be completed can be removed from the Sprint backlog (still, as long as they do not affect the Sprint Goal) to reflect the current known plan for the sprint; this however is not mandatory, as the items may remain in the Sprint Backlog if the team wishes to attempt their delivery despite the affected capacity loss.</p>

8.3 8.3 - Examples

- [8.3.1 Retrospective \(see page 427\)](#)
- [8.3.2 Product and Sprint Goal \(see page 433\)](#)

8.3.1 8.3.1 Retrospective

The purpose of the Sprint Retrospective is to self-reflect on the functioning of the Scrum Team, with a specific focus (but not limited to it) on the recently completed Sprint. The retrospective allows the Scrum Team to review their work in a *Transparent way*, *Inspecting* all positive and improvable aspects of the team, in order to incorporate the needed *Adaptations* into their future iterations.

Link to Retrospective description: [4.6 - The Sprint Retrospective \(see page 126\)](#)

This section will show you examples of how to conduct retrospectives in your team.

8.3.1.1 Tools for retro:

- **EasyRetro** - a web-based tool for a retrospective. Example Link: <https://easyretro.io/>
- **Retrium** - a web-based tool for a retrospective. Example Link: <https://www.retrium.com/>
- **Miro** - a web-based board where you can do custom retrospectives. Example Link: <https://miro.com/>

EG/NGA does not provide those tools. Most of them have free subscription plans.

Ideas for a retrospective :

Links to retrospective ideas, examples, and explanation:

Retromat: <https://retromat.org/en/?id=59-51-50-88-67>

EasyRetro resources with 100+ ideas: <https://easyretro.io/retrospective-ideas/>

8.3.1.1.1 Warm-up:

Mood check - easy way to check how the team feels on a day with retro. Basically, we are putting on a whiteboard or on web-based board 3 faces (Smaily, Neutral and Sad). Each person from the team is adding her or his name. When all team members will be on board Scrum Master and team is discussing their moods. Full exercise will take from 10 to 15 minutes.



One word - As a Scrum Master ask team members to describe the last sprint in one word. After that discuss each person's word. Full exercise will take from 10 to 15 minutes.



ESVP - Ask people to write down their feeling on a sticky note on the whiteboard or web-based board. After that please go through all team members to check why they are feeling as they marked in the exercise. Please also start a conversation during this exercise base on results marked by the team. As a Scrum Master and developers, you will get input from developers what you can change to help out the team members to feel better during retrospective event.

1. Explorer, exciting to be in the retro and eager to discover and learn new things.
2. Shopper, happy to be in the retro and open to learn new things.
3. Vacationer, glad to be away from his desk.
4. Prisoner totally doesn't like the retro and it is punishment (s)he needs to be here.



8.3.1.2 Insight gathering from the last sprint:

4 L (Liked, Learned, Lacked, and Longed For) - Ask people to write down the things that happened during the last iteration and put them in the specific category. When first part will be added ask people to group cards in those 4 categories. Present subjects to the team and discuss them.

liked	learned
longed for	lacked

Went Well - To Improve - Action Items - Basic approach to retrospective. Scrum Master is presenting a board with 3 columns where we have:

- Went Well - Scrum Team is putting all details regarding what was good, what they have achieved, or cudos into the board.
- To Improve - things that were not so good, impediments, problems in the sprint.
- Action Items - actions for next sprint to implement to work better.

Scrum team at the beginning is adding cards into Went Well and To Improve cards. After that, we are checking what is on the board and doing voting to set the order of discussion (number of votes please choose base on the number of members in the team). During discussion please start with Went Well column after that start conversation To Improve. Base on To Improve please put with the team action Items for the next Sprint.

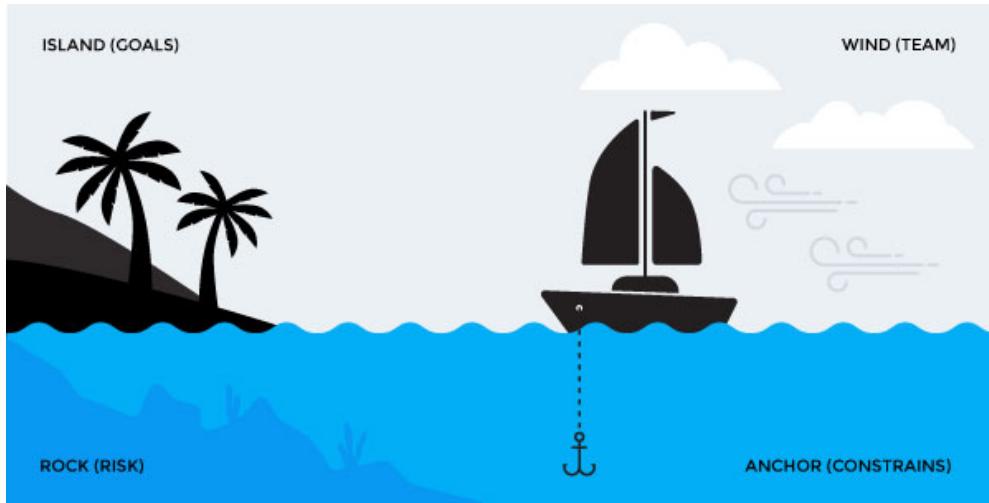
Went well	To improve	Action items

Sail Boat Retrospective - First, we draw a SailBoat, rocks, clouds, and a couple of islands like it is shown on the picture on a flip chart or web-based board.

- The islands represent teams' goals/visions. They work every day to achieve these islands.
- The rocks represent the risks they might encounter towards their vision.
- The anchor on the SailBoat is everything that is slowing them down on their journey.

- The clouds and the wind represent everything that is helping them to reach their goal.

After drawing please give the team 10-15 min to add their ideas on the board. After that please start brainstorming on each part of the picture. Base on the discussion please also put the action items for the next sprint.

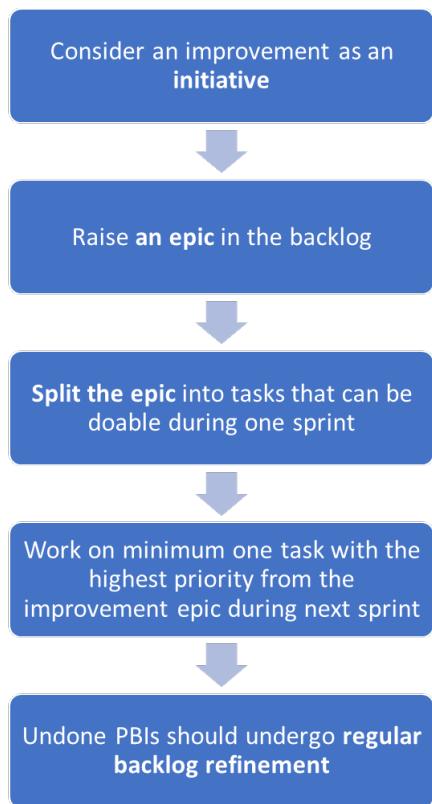


8.3.1.3 Deciding what to do:

8.3.1.3.1 How to deal with long-term retrospective improvements?

Sometimes during a retrospective, the team creates action points that cannot be satisfied during one sprint due to their size or complexity.

To support tracking long-term retrospective improvements and ensure they are correctly executed, one can follow below actions:



Above approach enables capturing the full scope of the long-term improvement and sends clear message when the improvement can be considered as implemented.

8.3.1.4 Closing Retrospective:

During the closing of the retrospective, we can check with the team if the team is satisfied with this event.

1. **Energy for the next sprint** - Ask the people to put a sticky note on the energy level that represents their eagerness for the next iteration.



2. **Final word** - Ask team about their one word regarding retrospective. A quick way of getting feedback.
3. **Retro Dart** - Ask the attendees to play "darts" with a sticky note. A quick way of getting feedback.



8.3.1.5 Five dysfunctions of a team - an alternative approach to retrospective

It is a concept based on Patrick Lencioni's book "Five dysfunctions of a team".

The aim of the workshop is to make team aware about its unique strengths and the areas where the team can improve.

It is recommended to conduct such workshop during a regular Retrospective when:

- We would like to improve team's performance and wellbeing
- The team looks for new ways to facilitate a Retrospective
- Last sprint was successful and there are not many topics to discuss

How to facilitate a workshop? (supportive files can be found below)

1. Present the survey to the team and explain it's goal
2. Conduct the survey
3. Analyze the results and decide which dysfunction should be talked through the most.
4. Present only the slides 2 and 4-8 explaining your observations on survey results
5. After presenting the content to the team, brainstorm actions leading to team's growth. Long-term actions and ideas are attached in Scrum Master's handbook in the presentation.

The survey:	 Survey.xlsx
Five dysfunction of a team - ppt	 5dysfunctions.pptx

8.3.2 Product and Sprint Goal

Full description/explanation regarding Sprint Goal and Product Goal can be found: [4.3 - The Sprint Planning \(see page 117\)](#). Below please find examples of them.

8.3.2.1 Product Goal

What is a Product Goal?

The Product Goal can be understood as the North Star to which the product advances in the long-term. It should be crafted by the Product Manager(s) in co-operation with the Product Owner(s). The Product Goal should be reflected in the Product Backlog.

Examples:

1. Expand Loan Offer for the Australian customers
2. Launch mobile application for pregnant women to inform them about their pregnancy
3. Launch web-based application for the doctors to help to treat cancer

8.3.2.2 Sprint Goal

What is the Sprint Goal?

The Sprint should have a Sprint Goal that relates to the Product Goal. The Sprint Planning event can be distinguished into 3 parts: a proposition of how the product can increase in value, deciding on what needs to be done, and how it could be done (task decomposition or task break-down)

Examples:

1. Prepare Credit Line for customers
2. Prepare Instalment Loan for customers
3. Prepare web-based credit line application form for customers - change from paper form to digital form.

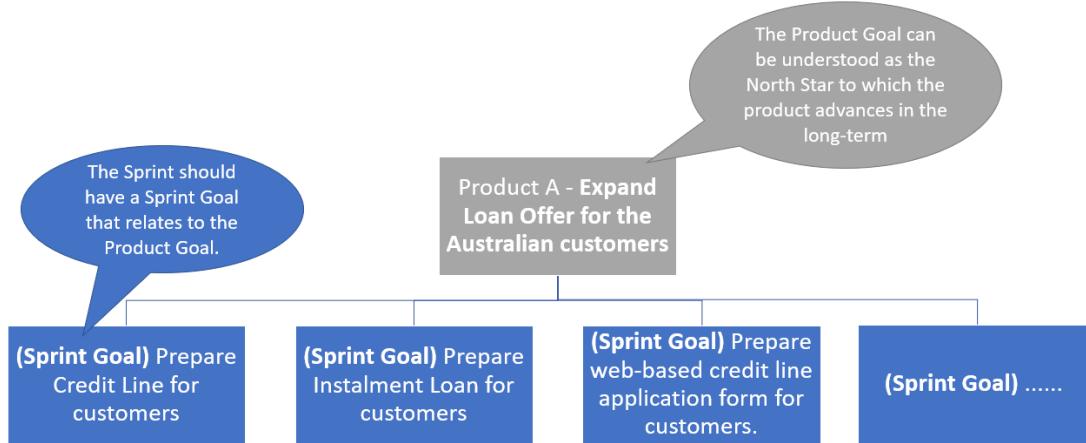
8.3.2.3 Correlation between Product Goal and Sprint Goal

Product Goal and Sprint goal are correlated with each other.

Examples:

1. **(Product Goal)** Expand Loan Offer for the Australian customers
 - a. **(Sprint Goal)** Prepare Credit Line for customers
 - b. **(Sprint Goal)** Prepare Instalment Loan for customers
 - c. **(Sprint Goal)** Prepare web-based credit line application form for customers - change from paper form to digital form.
 - d. **(Sprint Goal)**
2. **(Product Goal)** Launch web-based application for the doctors to help treat cancer
 - a. **(Sprint Goal)** Buildability to add x-ray picture into the system
 - b. **(Sprint Goal)** Build an interface for messaging other doctors
 - c. **(Sprint Goal)** Enabling CT scans to be sent to other providers through API - faster doctors' opinion/description
 - d. **(Sprint Goal)**

Visualization:



8.3.2.4 How to craft Sprint Goal and Product Goal:

8.3.2.4.1 Product Goal

One of the techniques which we can use to set the Product Goal is the usage of SMART

S = specific, significant, stretching

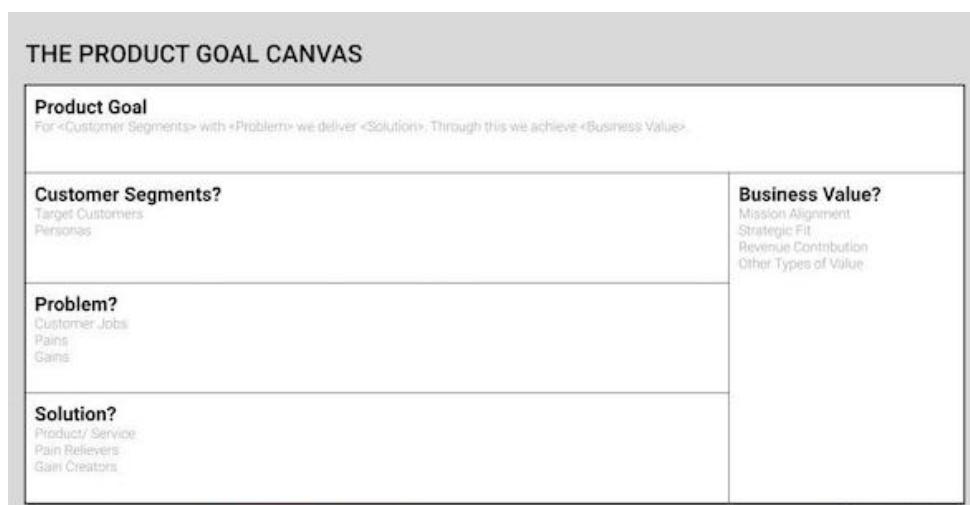
M = measurable, meaningful, motivational

A = agreed upon, attainable, achievable, acceptable, action-oriented

R = realistic, relevant, reasonable, rewarding, results-oriented

T = time-based, time-bound, timely, tangible, trackable

The second which we can use is the Product Canvas:



Customer Segments - Every product team and their stakeholders need to be aligned 100% on which customer segments they want to serve. Tools like the Empathy Map Canvas or concepts like Personas can be very helpful in order to define the customer segments and detail them. In many cases, a product serves more than one customer segment. It is OK to include multiple customer segments in your Product Goal Canvas.

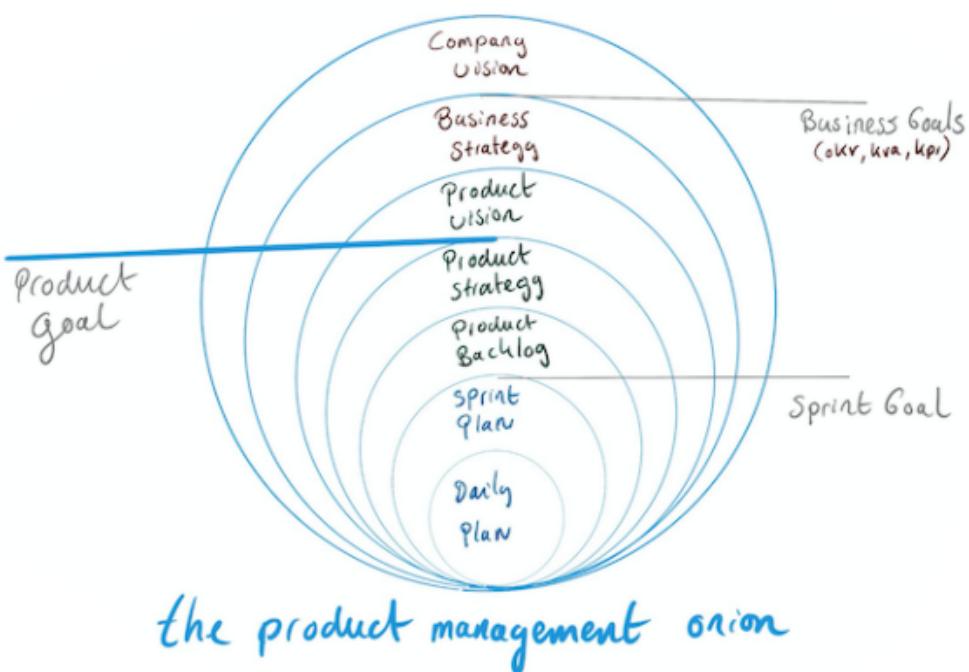
Problem - Once a team has clarity on the customer segment they want to serve, they need to align on the problem they intend to solve. A problem could be a specific job-to-be-done, customer pains, and/or gains.

Solution - This building block of the Product Goal Canvas summarizes the Solution the team wants to implement. The team needs to state what the product or service is going to be without going too deep into the details. The unique value proposition of the product e.g. pain relievers or gain creators can and potentially should be mentioned.

Business Value - This building block captures the desired outcomes for the organization itself. Why are we as an organization building this product? The first three building blocks are all centred around our customers and their needs. This last one is about us. Is this product going to help us advance our mission? Does it deliver toward our strategy? How will measure its success? Will it contribute revenue directly or indirectly?

The third technic is Product Onion.

The planning onion is a great way for product managers and teams to structure and frame different levels of planning. These levels each require different cadences of reviewing and updating, and each has different people that it should be serving.



- Set Product Goals together, leveraging the collective knowledge of the group but also as a first step towards acceptance and sharing of the goals. If you find this difficult to facilitate, have a look at liberating structures.

- Make sure you communicate the Product Vision before talking about goals, make sure we all have the same North Star.
- Next focus the discussion using “the lens of the product manager”. You don’t need the exact format listed below, but make sure you take an outside-in view of the goals you are setting.

For Product Onion you can refer to the links below:

<https://www.scrum.org/resources/blog/product-goal>

<https://www.romanpichler.com/blog/product-goals-in-scrum/>

8.3.2.4.2 Sprint Goal

The Sprint Goal is the single objective for the Sprint. Although the Sprint Goal is a commitment by the Developers, it provides flexibility in terms of the exact work needed to achieve it. The Sprint Goal also creates coherence and focus, encouraging the Scrum Team to work together rather than on separate initiatives.

Steps to set the Sprint Goal:

- The Product Owner presents the ordered backlog items to the team.
- The Scrum Team discusses and understands the work for this Sprint.
- Scrum Team forecasts and commits to the items that can be done.
- The Scrum Team creates the Sprint Goal for this Sprint

8.3.2.4.3 Q&A regarding Spring Goal

1. Sprint contains only bugs how to set Sprint Goal? - in this case, crafting sprint goal is very hard. But we can think about this in a different way. So we can establish Sprint Goal which will represent improvements for feature, system, part of the system by fixing the bug.
2. If developers are not working on one common increment? - in this case, we can set not 1 by 2-3 or even more sprint goals for the sprint. Those sprint goal will represent the personal developer goal for iteration.
3. What to do in case if the Sprint Goal will change? - in this case, developers can negotiate scope with the Product Owner or Product Owner with Developers can close the sprint before the end day of it and start a new one with a new Sprit Goal.

8.4 8.4 - EG Agile Certification program

8.4.1 Table of Contents



8.4.2 What is the program goal?

The EG Agile Certification program has been created in order to support EG employees in their voluntary learning journey and preparation to **acquire market-recognized agile certifications**.

Thanks to that we will be able to benefit from:

- **Strengthening the agile maturity** across the EG business units and promoting agile champions to become our ambassadors
- Expanding the **EG brand** as a workplace for agile-certified professionals
- Supporting EG's strategy that focuses on **Agile** product delivery& software development



8.4.3 Who is the program for?

The program is dedicated to all **EG employees** and especially those with the following roles & job functions:

- product managers & product owners
- scrum masters / flow masters / facilitators



8.4.4 What are the prerequisites to join the program?

In order to be qualified for the program, please fulfill the items below:

- completed NGA onboarding to EG software development (if applicable*)
- completed dedicated NGA training in Talentsoft
- completed PO/SM learning path training levels basic & advanced in Talentsoft for product owner and scrum master certifications
- approval from the manager to comply with program rules
- approval from the manager to approach particular certification exam on Scrum.org

Please contact your Agile Coach if you need access to the listed trainings in Talentsoft.

**Applicable for the employees who are subject to NGA software development onboarding*



8.4.5 What are the available certifications (valid for 2023)?

8.4.5.1 Professional Scrum Master I

People who have passed PSM I, achieving certification, demonstrate a fundamental level of Scrum mastery. PSM I certificate holders prove that they understand Scrum as described in the Scrum Guide and how to apply Scrum in Scrum Teams. PSM I holders have a consistent terminology and approach to Scrum. More information can be found here: [Professional Scrum Master™ I | Scrum.org⁹¹](https://www.scrum.org/professional-scrum-master-i)

8.4.5.2 Professional Scrum Product Owner I

People that have passed PSPO I and achieved certification demonstrate a fundamental understanding of the Scrum framework, and how to apply it to maximize the value delivered with a product. They exhibit a dedication to continued professional development, and a high level of commitment to their field of practice. Achieving PSPO I is the minimum demonstration of knowledge any Professional Scrum Product Owner should be able to make. More information can be found here: [Professional Scrum Product Owner™ I | Scrum.org⁹²](https://www.scrum.org/professional-scrum-product-owner-i)



8.4.6 How does the funding look like?

Estimated cost of exam is 150-200\$ depending on certification type.

The certification exam fee will be covered by the participant, however the fee will be **refunded by EG** under the following condition:

- the **exam is passed** by the participant **within a month** of completing the course.



8.4.7 Course information:

The format of the course will consist of **ten 1-hour group sessions** with an Agile Coach over a period of **3 months**, called training cycles. There will be approximately 2-3 training cycles per year, however this can be modified depending on the demand and interest.

The course will be split into a general part for all certifications (~70%), and a specialized part (~30%) dedicated to each certification, where participants will be split into sub-groups.

The group sessions will be held in the working time. Homework tasks that should be completed in private time may also appear.

⁹¹ <https://www.scrum.org/professional-scrum-master-i-certification>

⁹² <https://www.scrum.org/professional-scrum-product-owner-i-certification>

It is highly recommended to consider homework tasks as a part of the course and conduct self-learning activities outside of the group sessions in order to successfully complete the certification program.

The content of the course will focus on:

- expanding the knowledge gained in the Talentsoft trainings with focus on certification scope
- conducting trial examinations and discussion on the reasonings behind the correct answers
- in-depth study of the Scrum Guide
- mutual support, community-like networking format to support each other in preparation for the exam

A single course cycle will be conducted by 1 dedicated Agile Coach.



8.4.8 Class Schedule

The third iteration of the program will consist of a single group running in accordance with the following schedule:

	PSM	PSPO	Session dates
1	x	x	📅 21 Sep 2023 13:00 - 14:00 CET
2	x	x	📅 28 Sep 2023 13:30 - 14:30 CET
3	x	x	📅 05 Oct 2023 13:00 - 14:00 CET
4	x	x	📅 12 Oct 2023 13:00 - 14:00 CET
5	x	x	📅 26 Oct 2023 13:00 - 14:00 CET
6	x	x	📅 09 Nov 2023 13:00 - 14:00 CET
7	x	x	📅 16 Nov 2023 13:00 - 14:00 CET
8	x		📅 22 Nov 2023 14:00 - 15:00 CET

	PSM	PSPO	Session dates
9		x	 23 Nov 2023 13:00 - 14:00 CET
10	x		 29 Nov 2023 14:00 - 15:00 CET
11		x	 30 Nov 2023 13:00 - 14:00 CET
12	x	x	 07 Dec 2023 13:00 - 14:00 CET



8.4.9 How can I sign up?

Participants will be able to sign up to the course via Talentsoft. The course group will be limited to 20 participants.

Signing up for the second iteration will be available starting September and the course will commence in the second half of September 2023.



8.4.10 Course plan and useful materials

Exam overview and best practices	 NGA certification program.pdf
----------------------------------	--

Agenda for particular classes

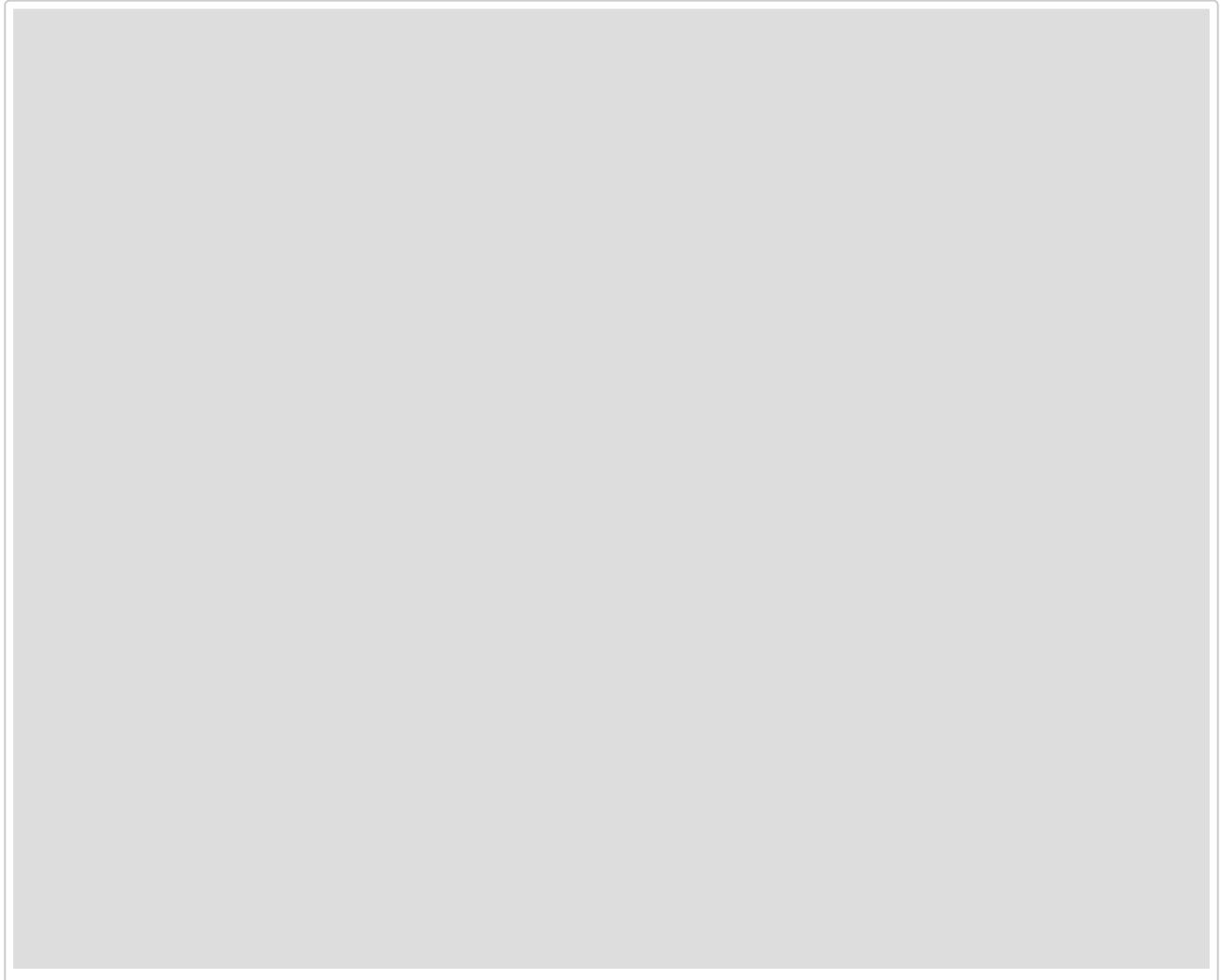


NGA certification...ram - agendas.pdf

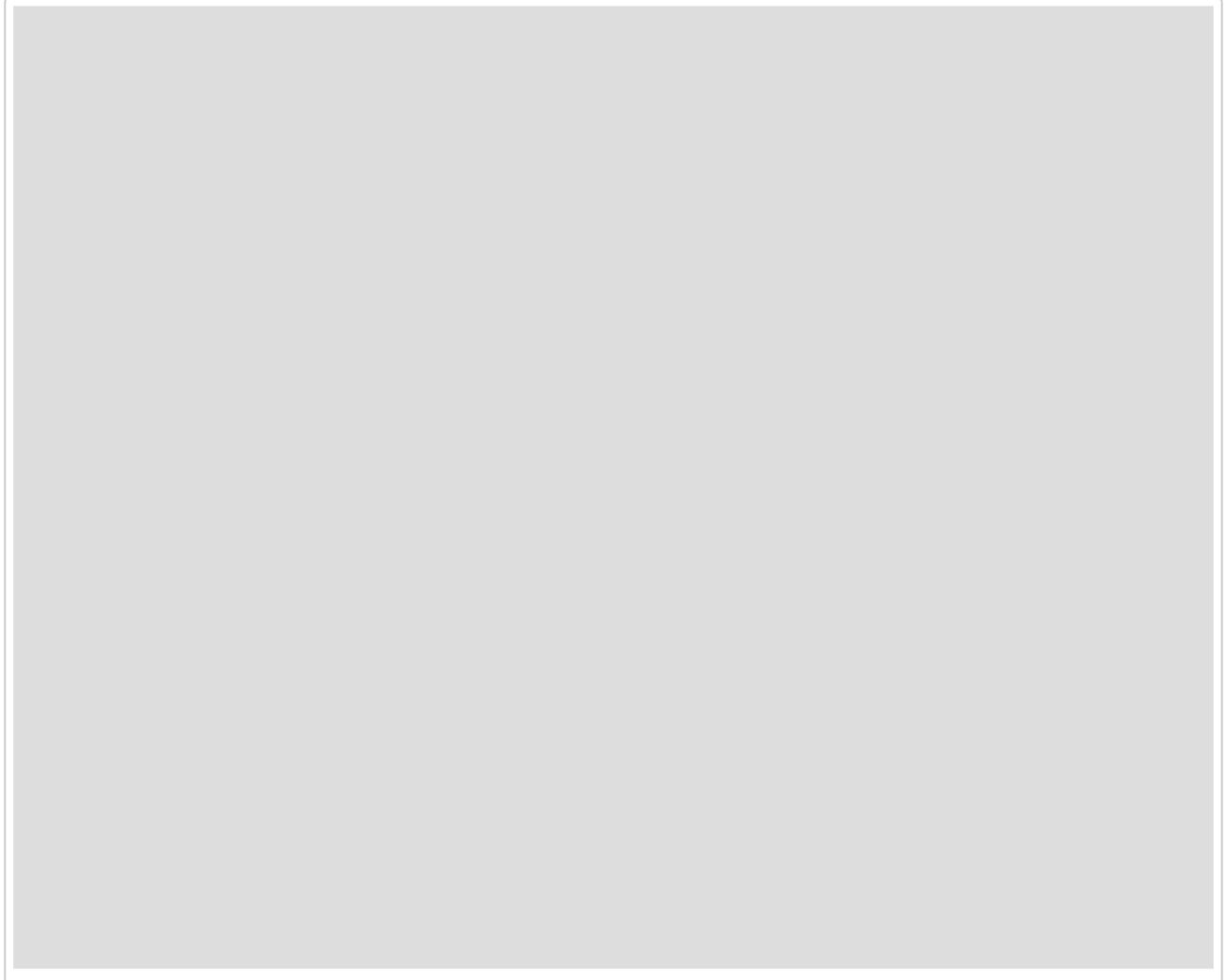
9 9 - Release notes

- NGA(i) - version 3.0 (see page 444)
 - NGA - version 1.1 (see page 445)
 - NGA - version 1.2 (see page 446)
 - NGA - version 1.3 (see page 447)
 - NGA - version 1.3.1 (see page 448)
 - NGA - version 1.4 (see page 449)
 - NGA - version 1.5 (see page 450)
 - NGA - version 1.6 (see page 451)
 - NGA - version 1.7 (see page 452)
 - NGA - version 1.8 (see page 452)
 - NGA - version 1.9 (see page 453)
 - NGA - version 1.10 (see page 454)
 - NGA - version 2.0 (see page 455)
 - NGA - version 2.1 (see page 456)
 - NGA - version 2.2 (see page 457)
 - NGA - version 2.3 (see page 458)
 - NGA - version 2.4 (see page 459)
 - NGA - version 2.4.1 (see page 460)
 - NGA - version 2.5 (see page 461)
 - NGA - version 2.6 (see page 462)
 - NGA - version 2.7 (see page 463)
 - NGA - version 2.8 (see page 464)
 - NGA - version 2.9 (see page 465)
 - NGA - version 2.10 (see page 466)
 - NGA - version 2.11 (see page 467)
 - NGA - version 3.1 (see page 468)
 - NGA - version 3.2 (see page 469)
-

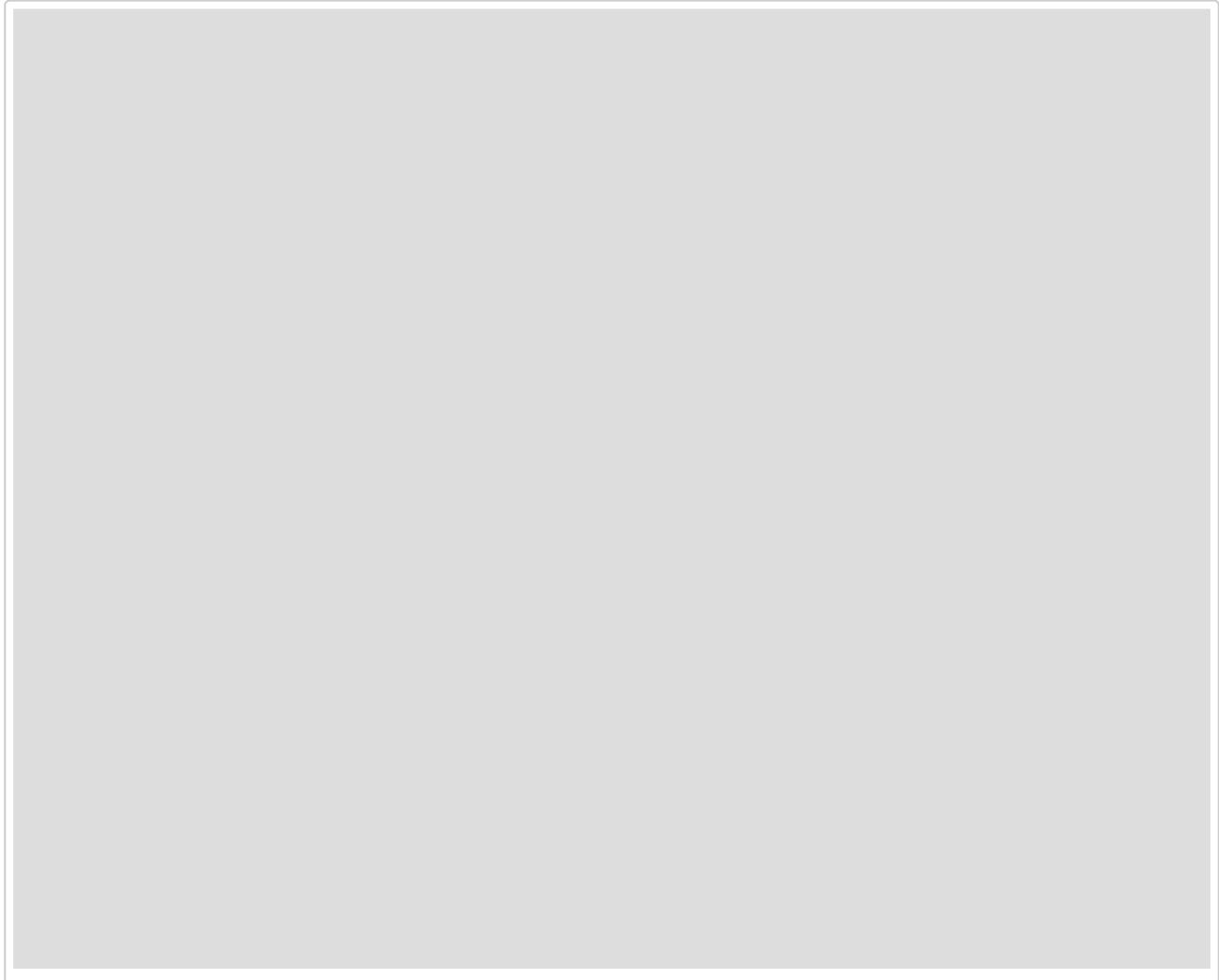
9.1 NGA(i) - version 3.0



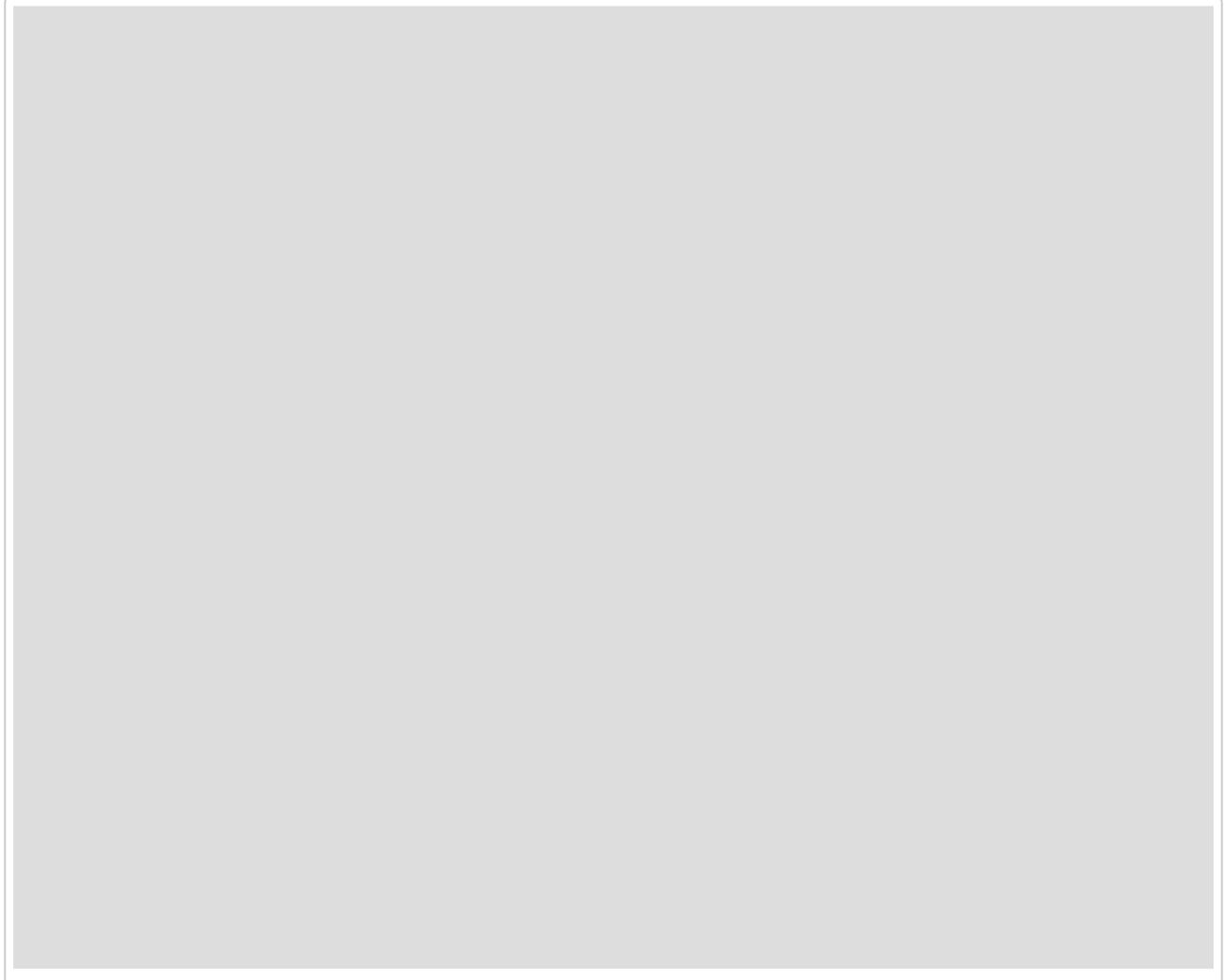
9.2 NGA - version 1.1



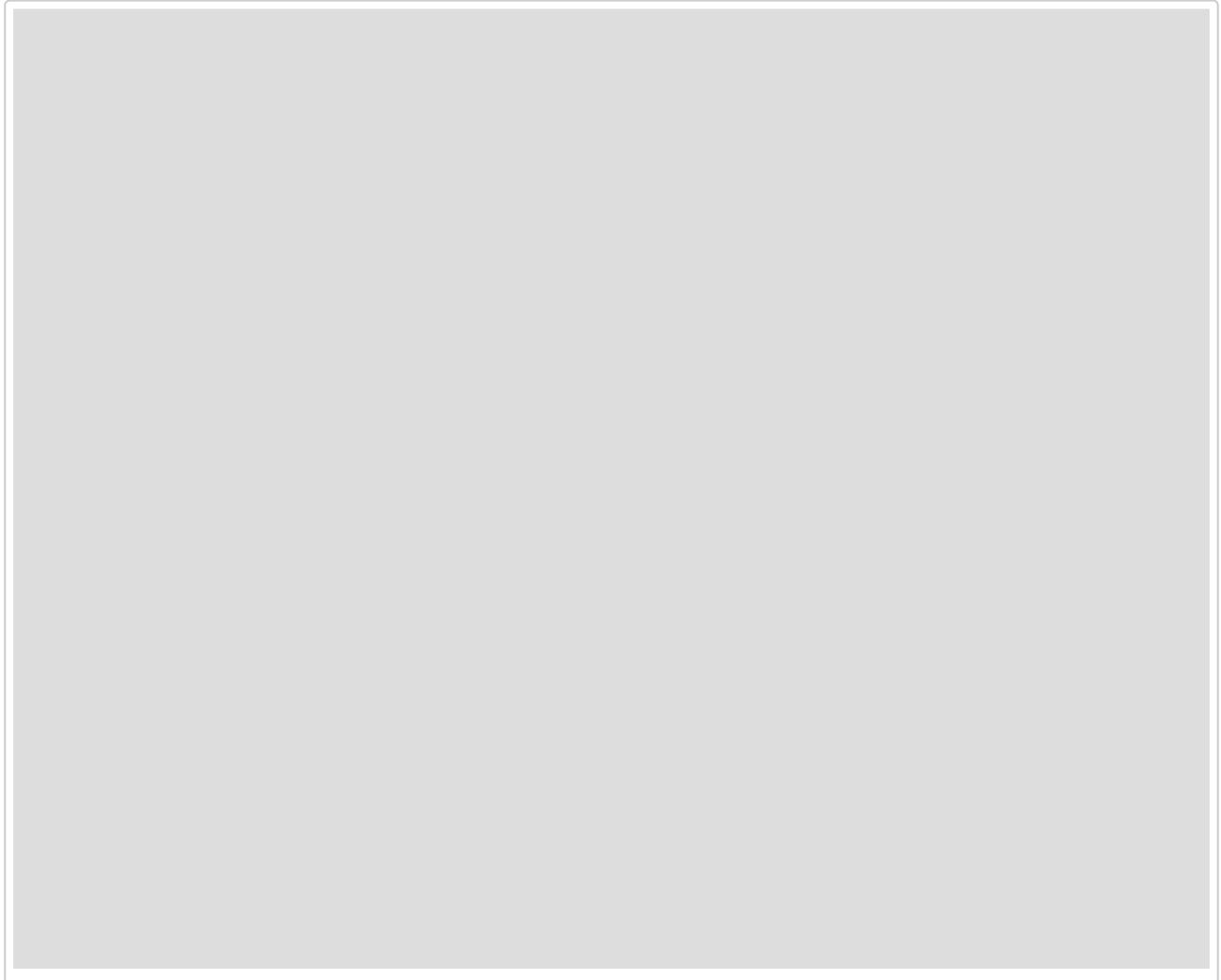
9.3 NGA - version 1.2



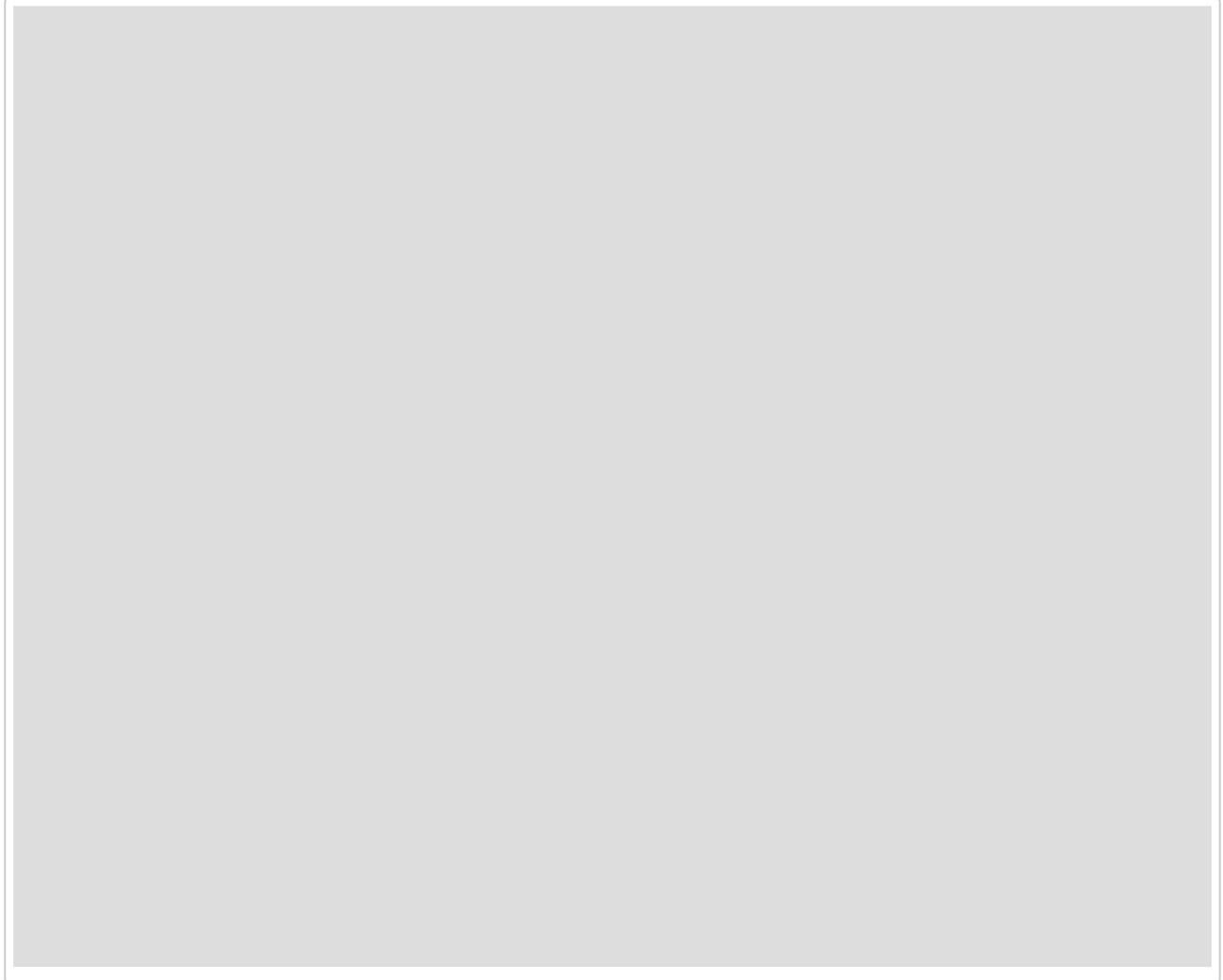
9.4 NGA - version 1.3



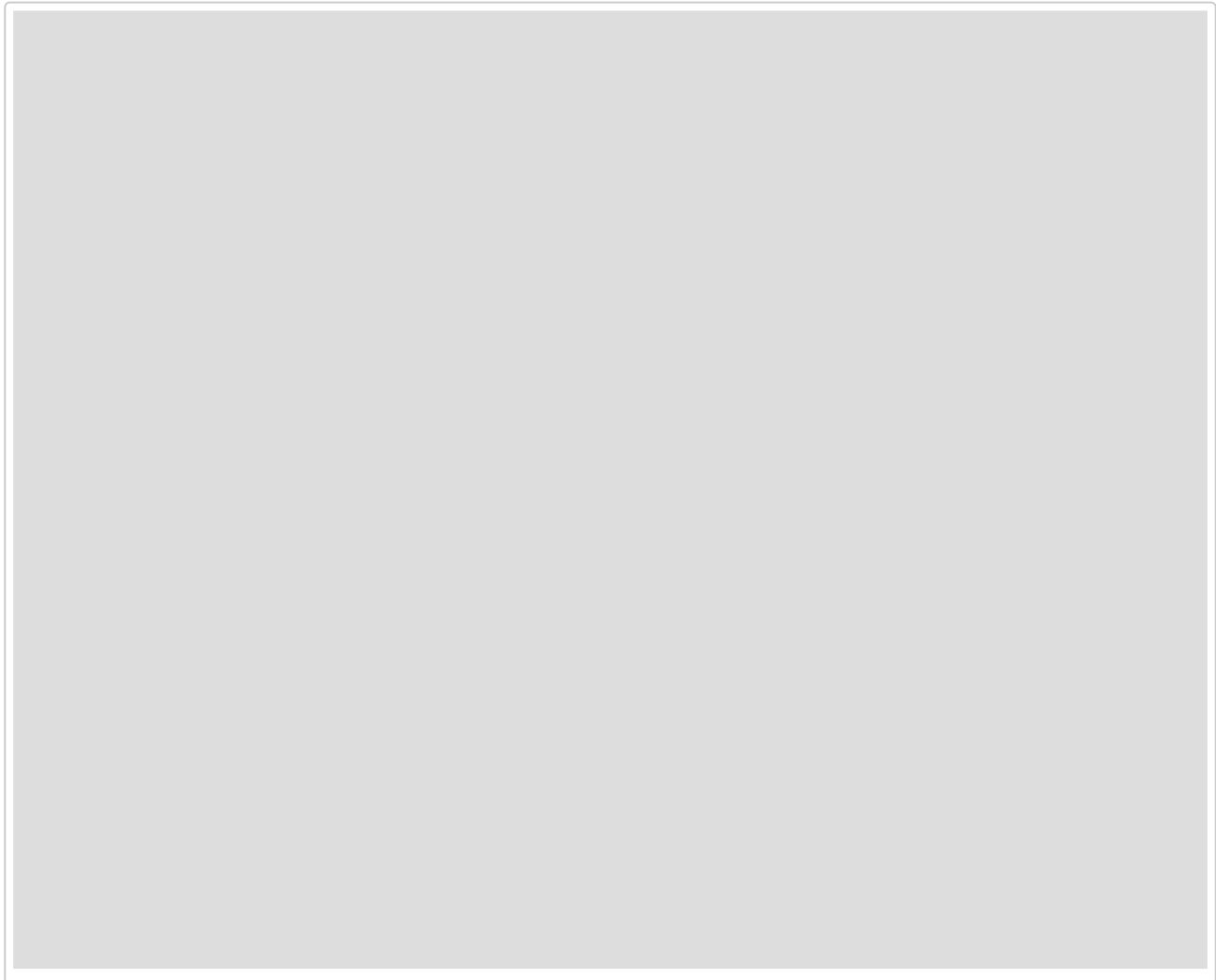
9.5 NGA - version 1.3.1



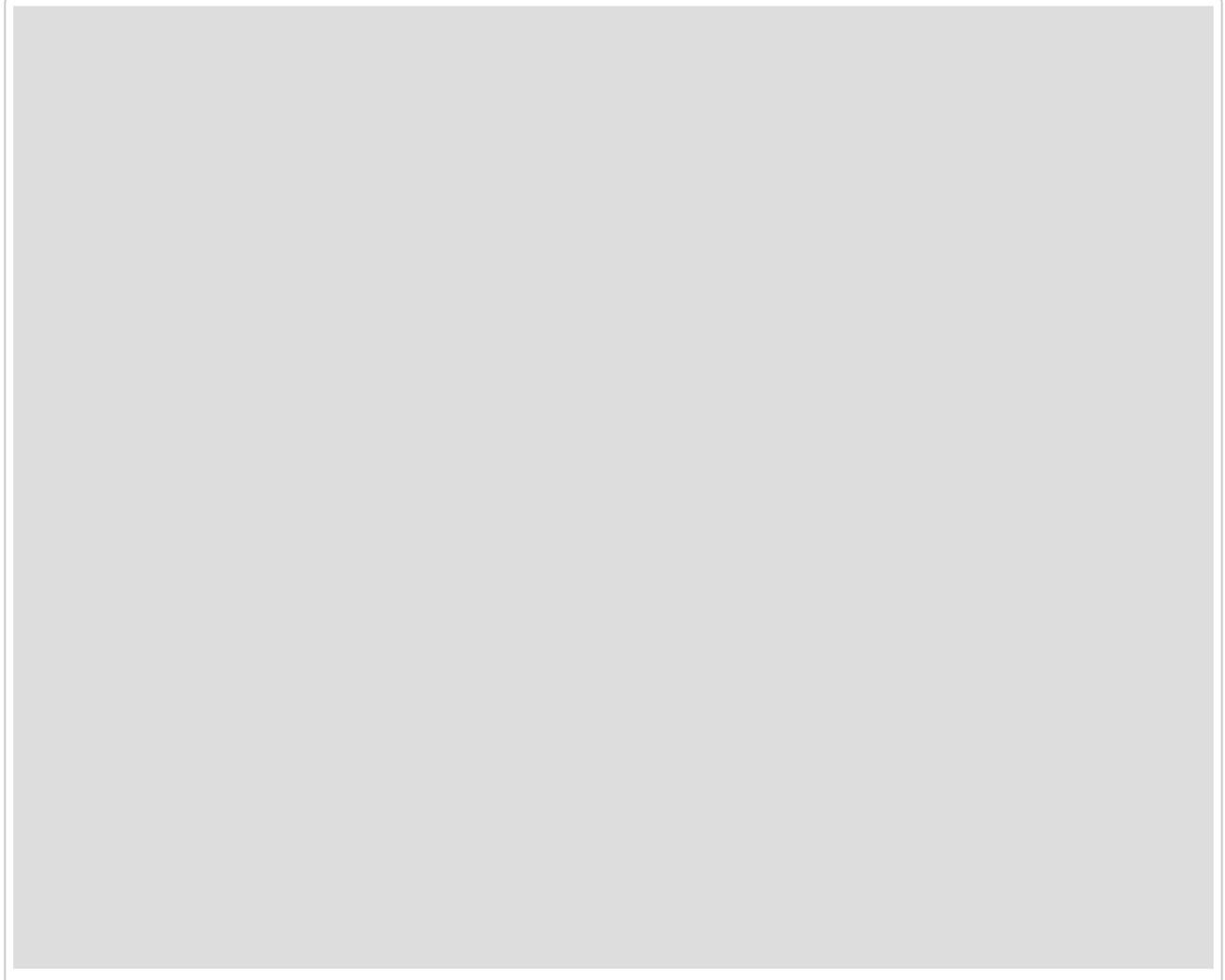
9.6 NGA - version 1.4



9.7 NGA - version 1.5



9.8 NGA - version 1.6



9.9 NGA - version 1.7



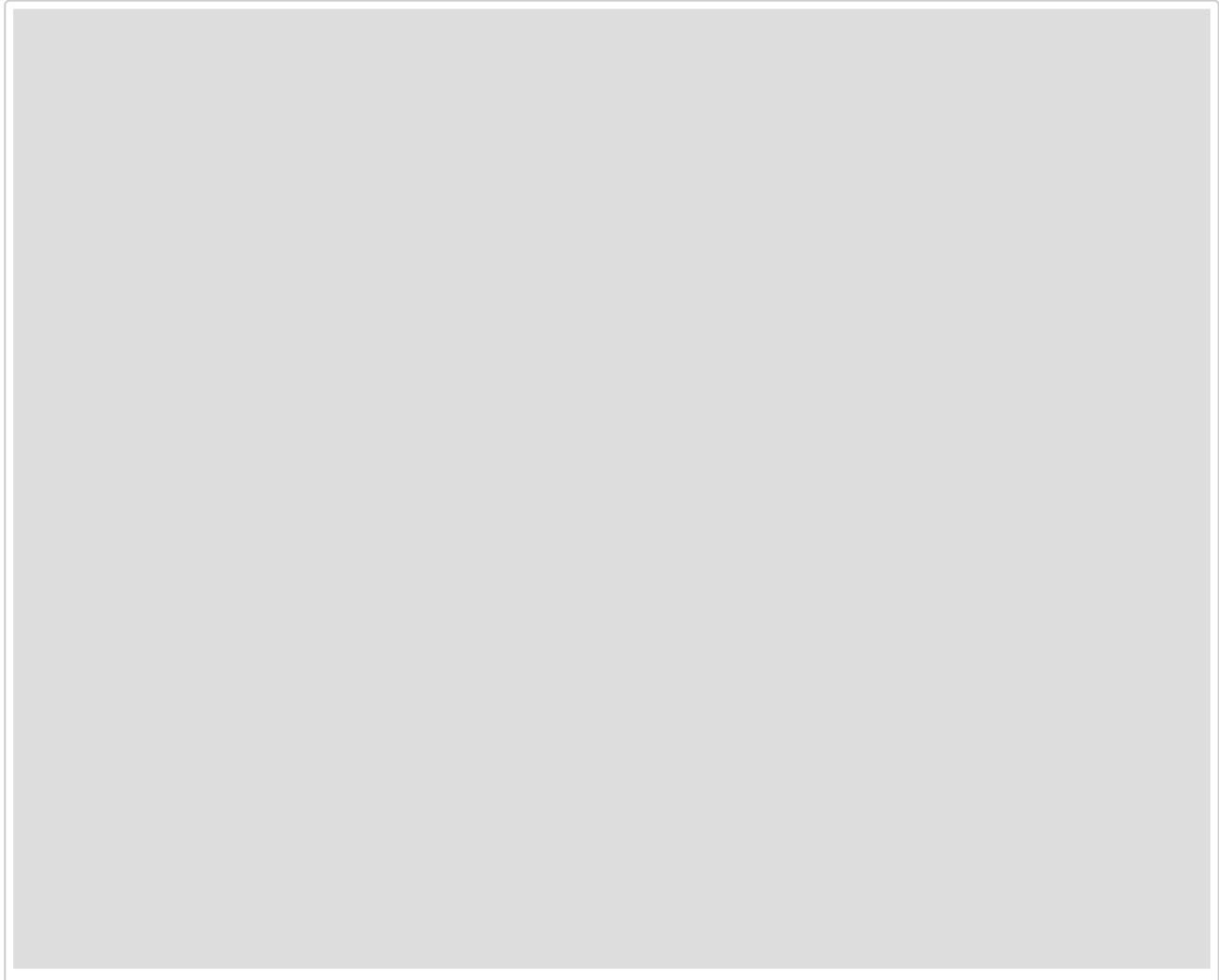
Release notes - version NGA...cf88221-050321-0806-448.pdf

9.10 NGA - version 1.8

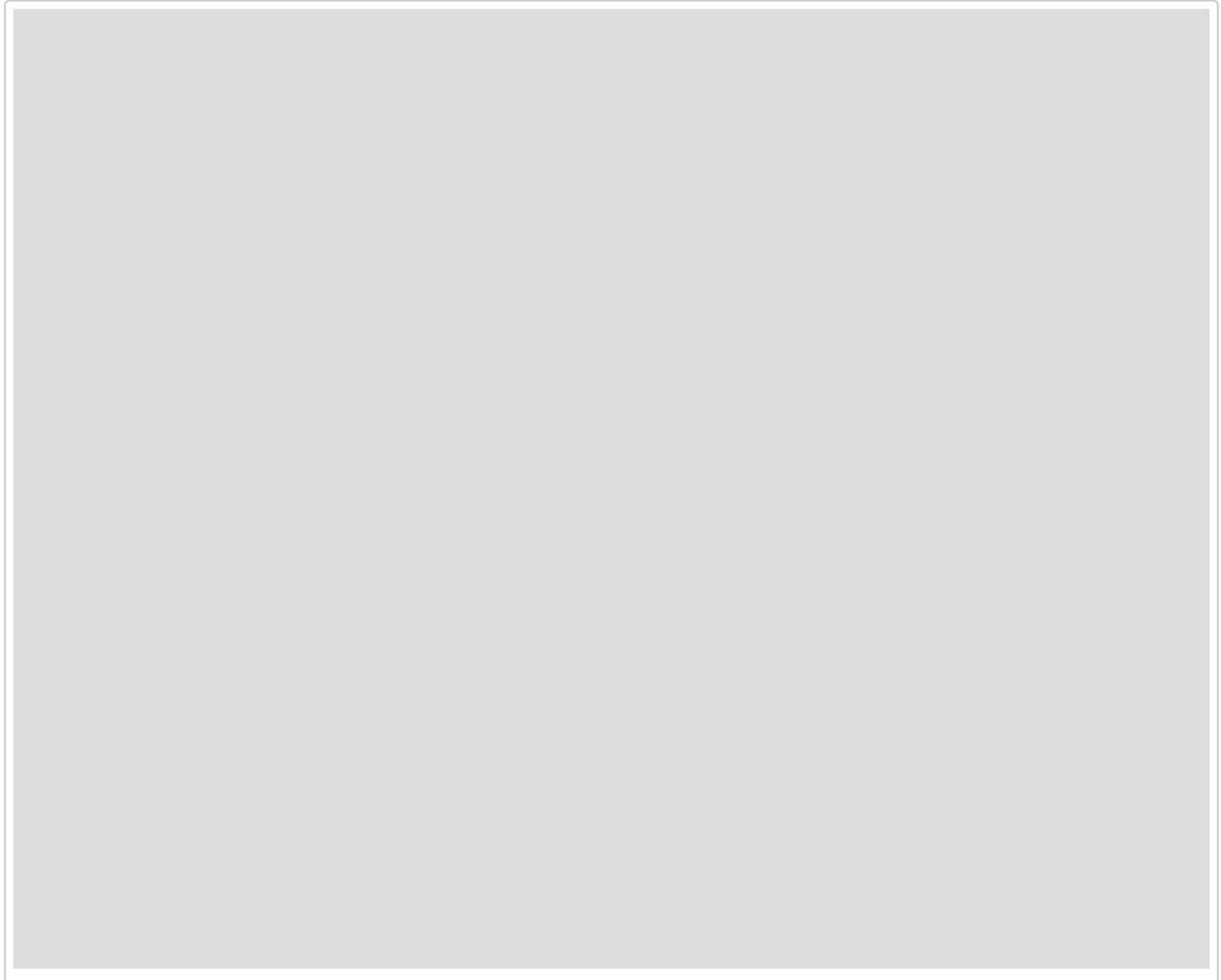


Release notes - v...521-0915-1620.pdf

9.11 NGA - version 1.9



9.12 NGA - version 1.10

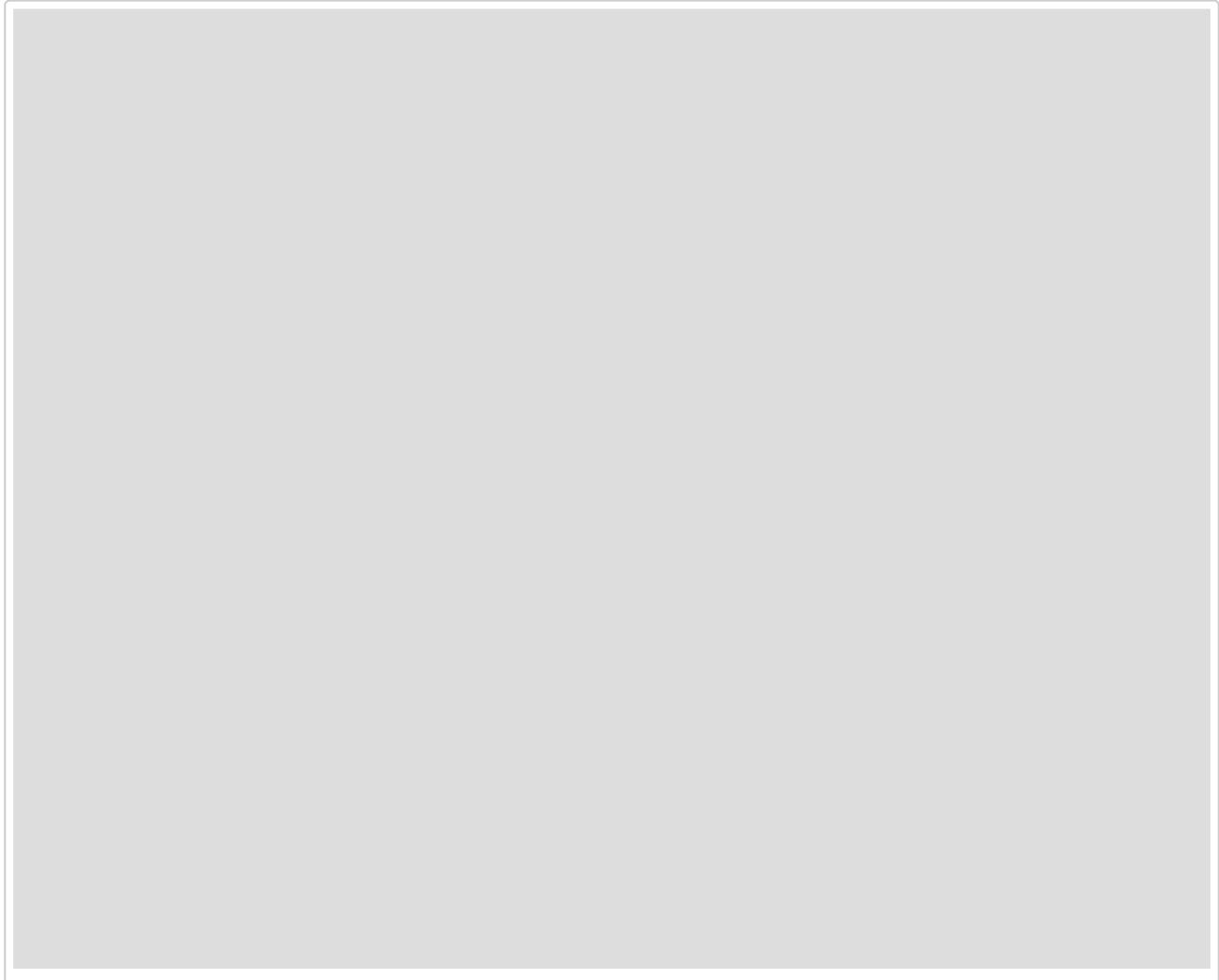


9.13 NGA - version 2.0

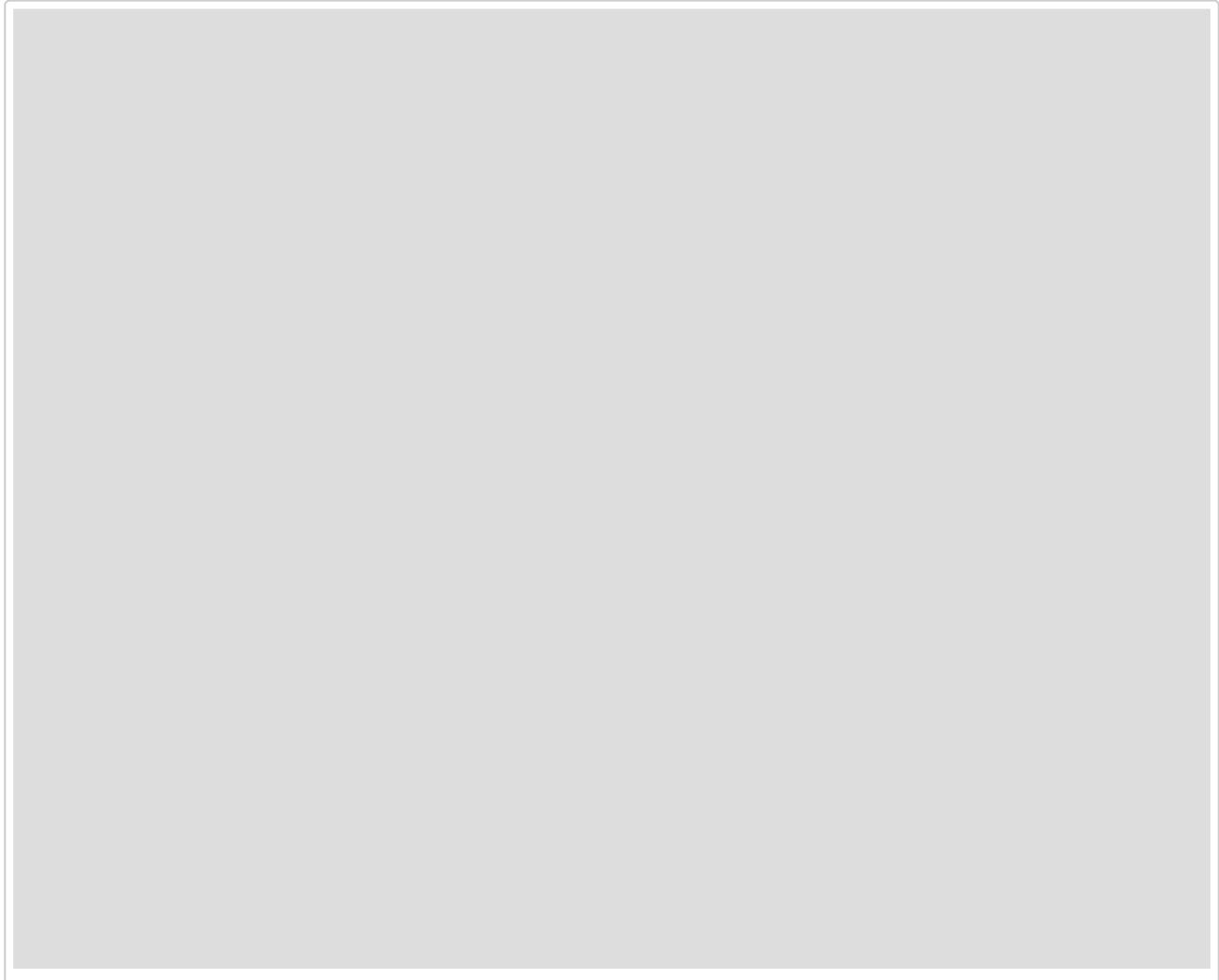


Release notes - version NGA ...file - Confluence EG A_S.pdf

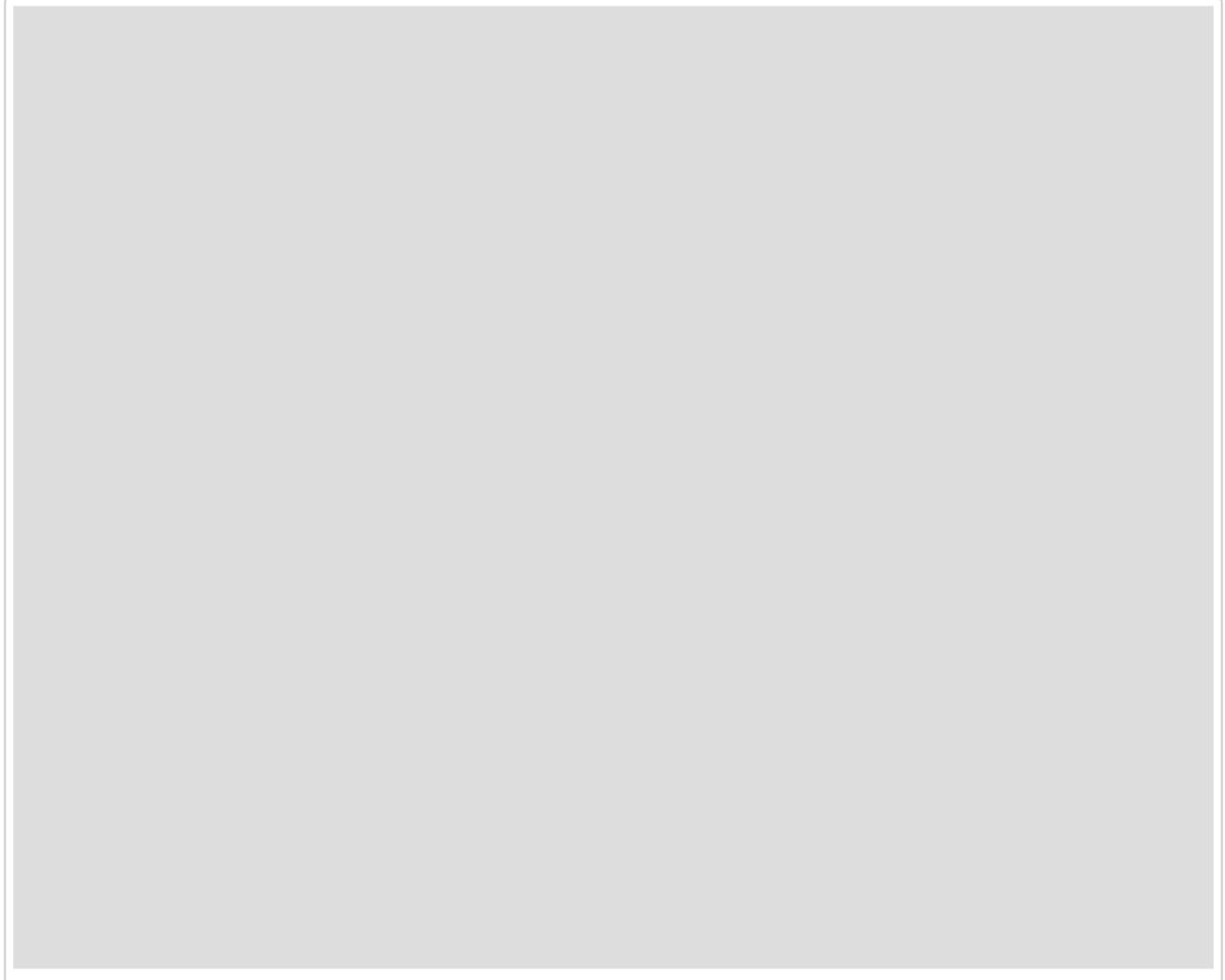
9.14 NGA - version 2.1



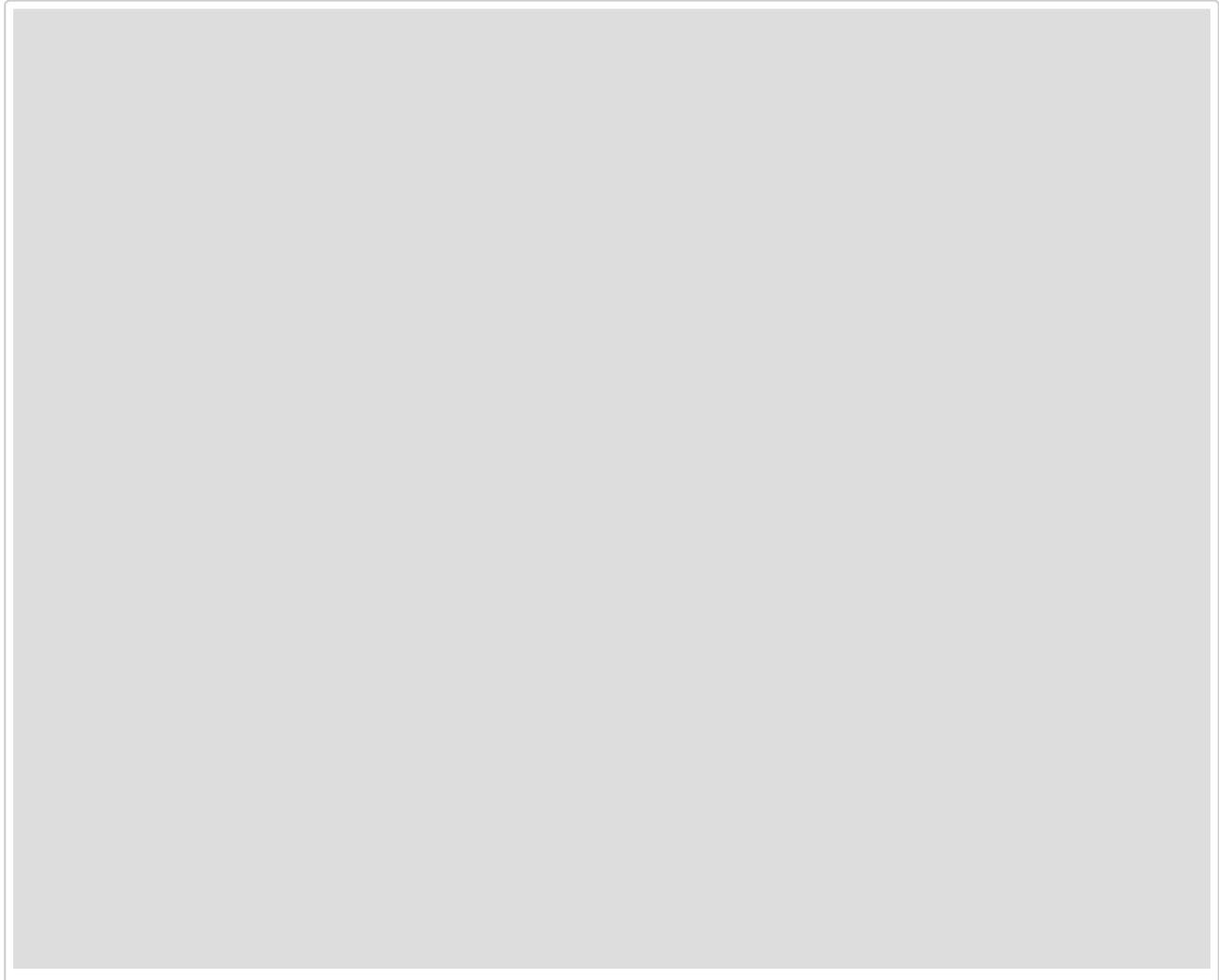
9.15 NGA - version 2.2



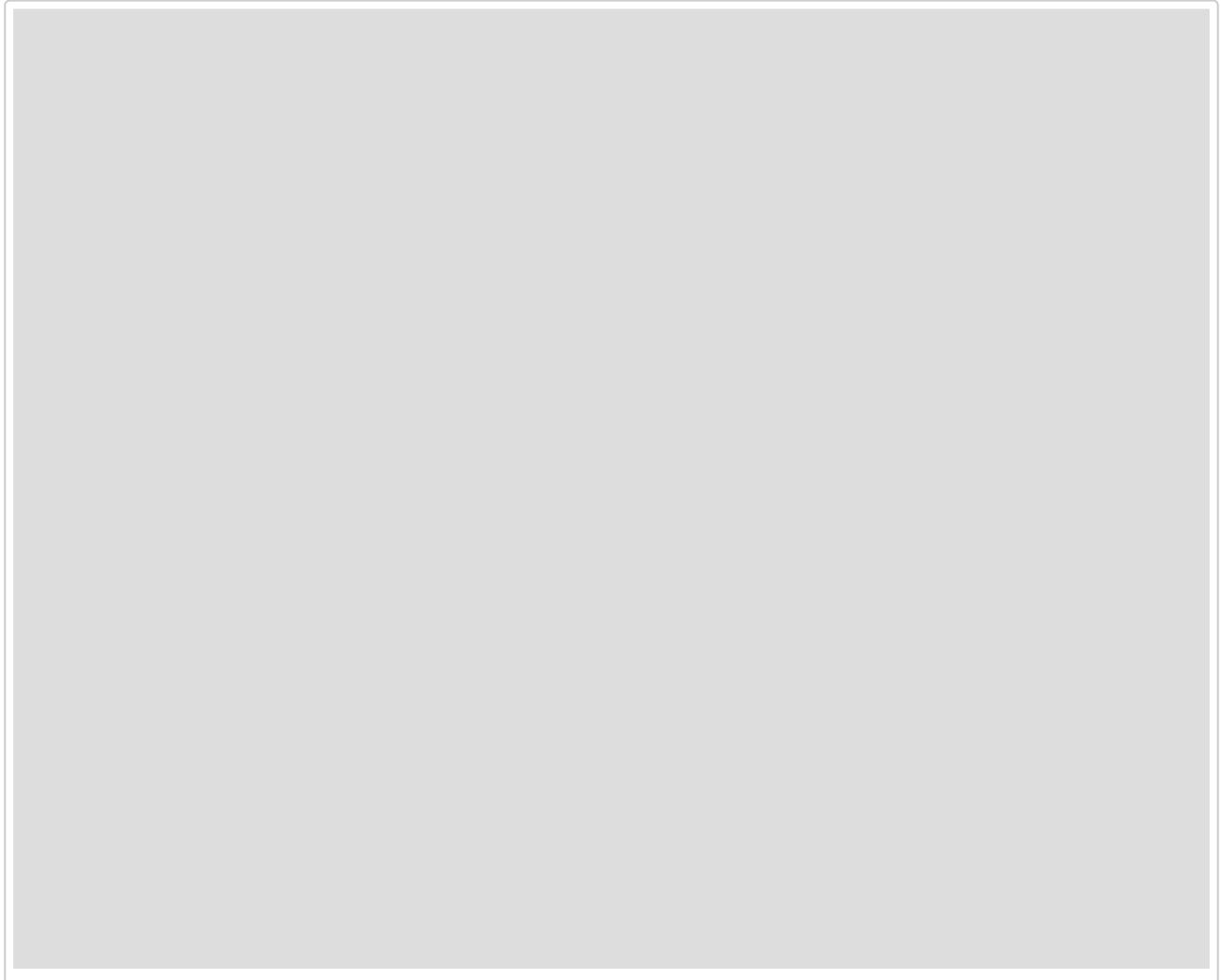
9.16 NGA - version 2.3



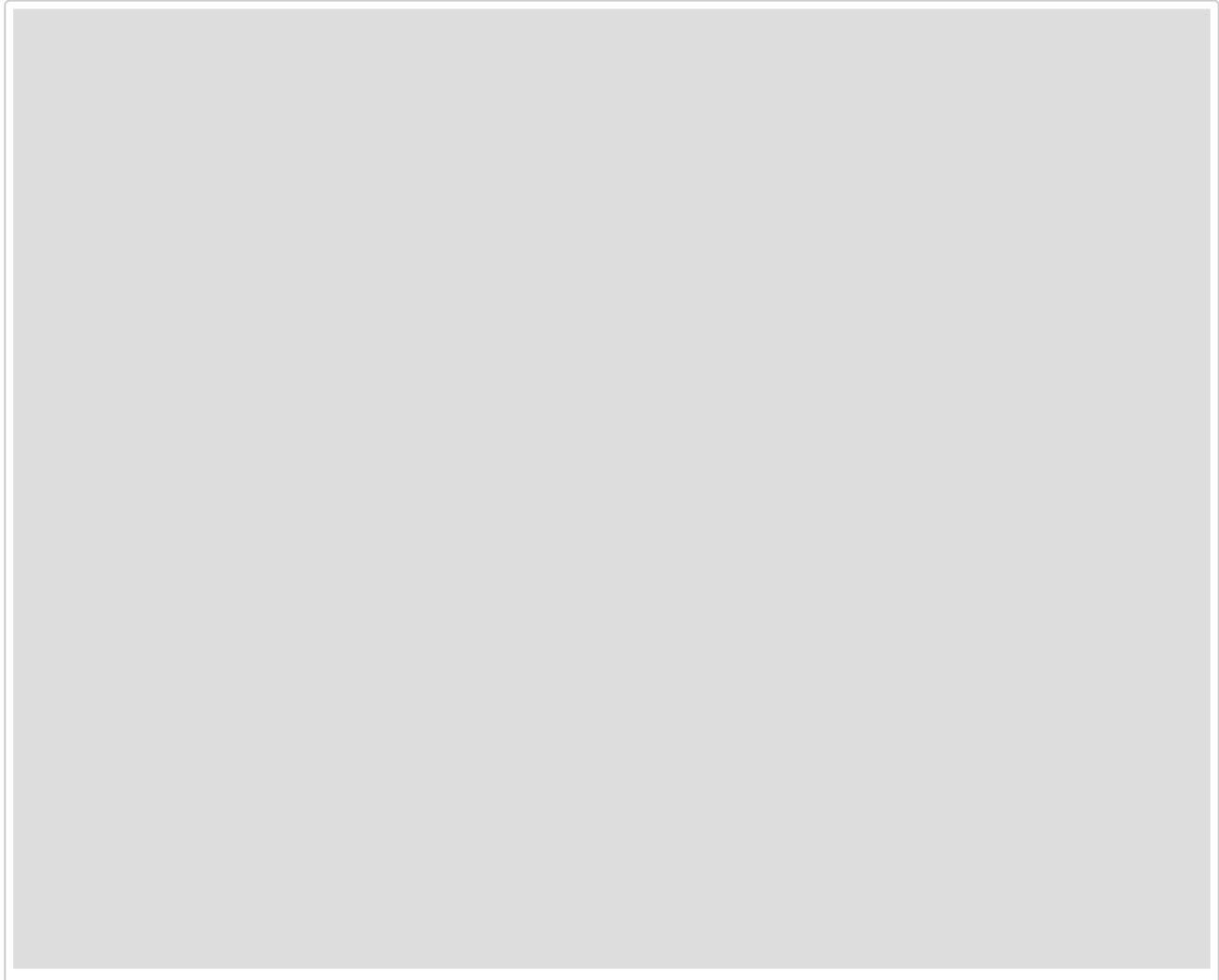
9.17 NGA - version 2.4



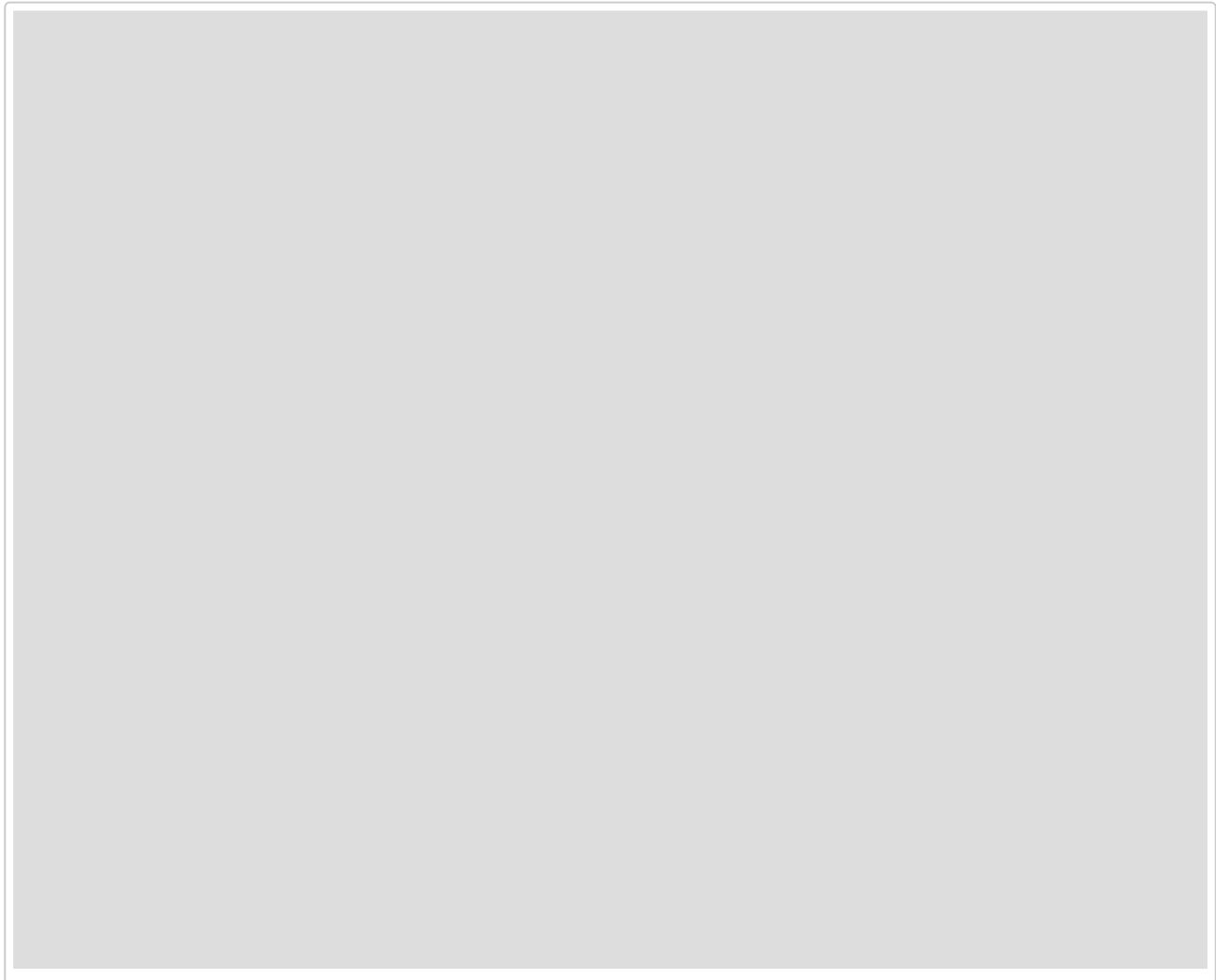
9.18 NGA - version 2.4.1



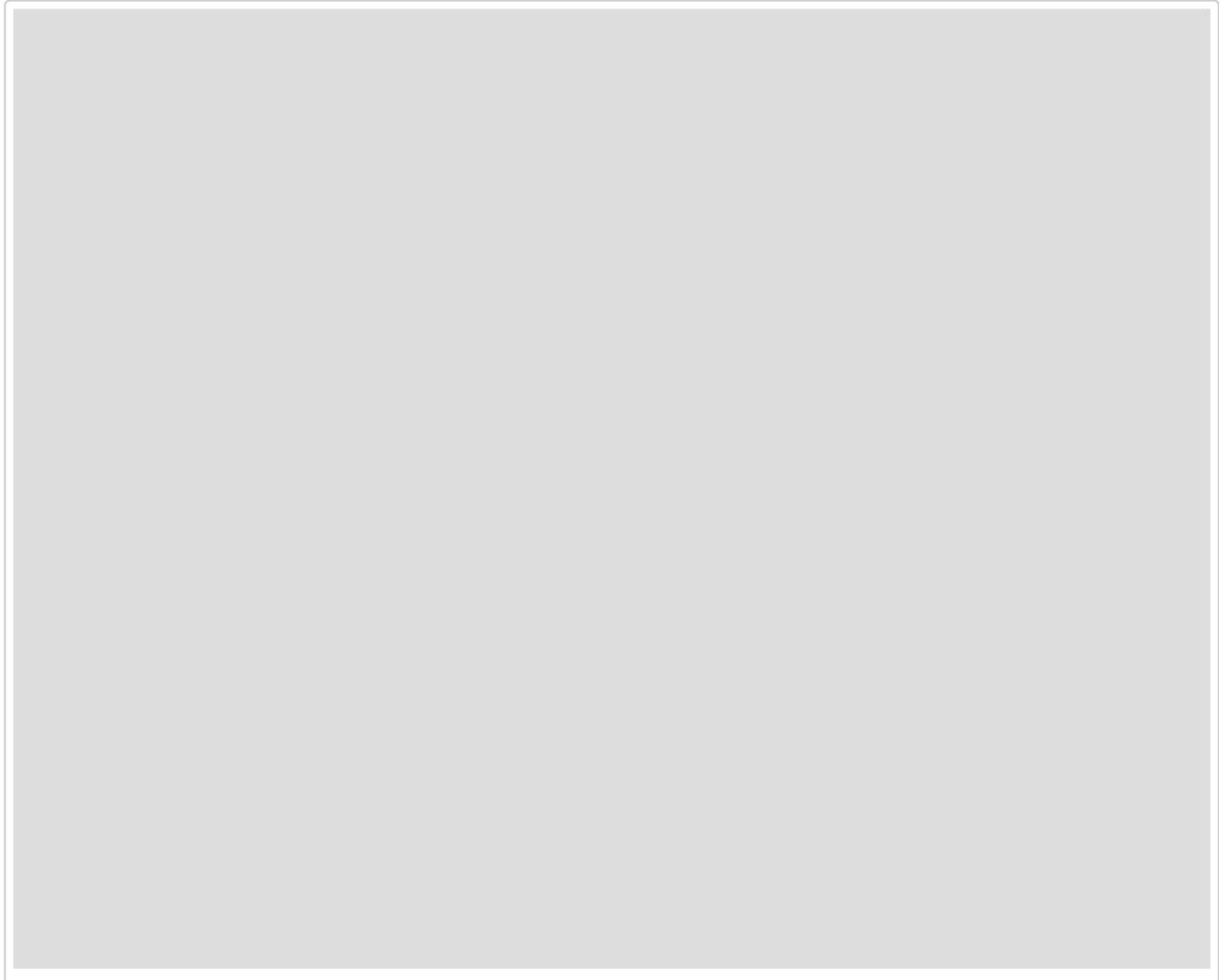
9.19 NGA - version 2.5



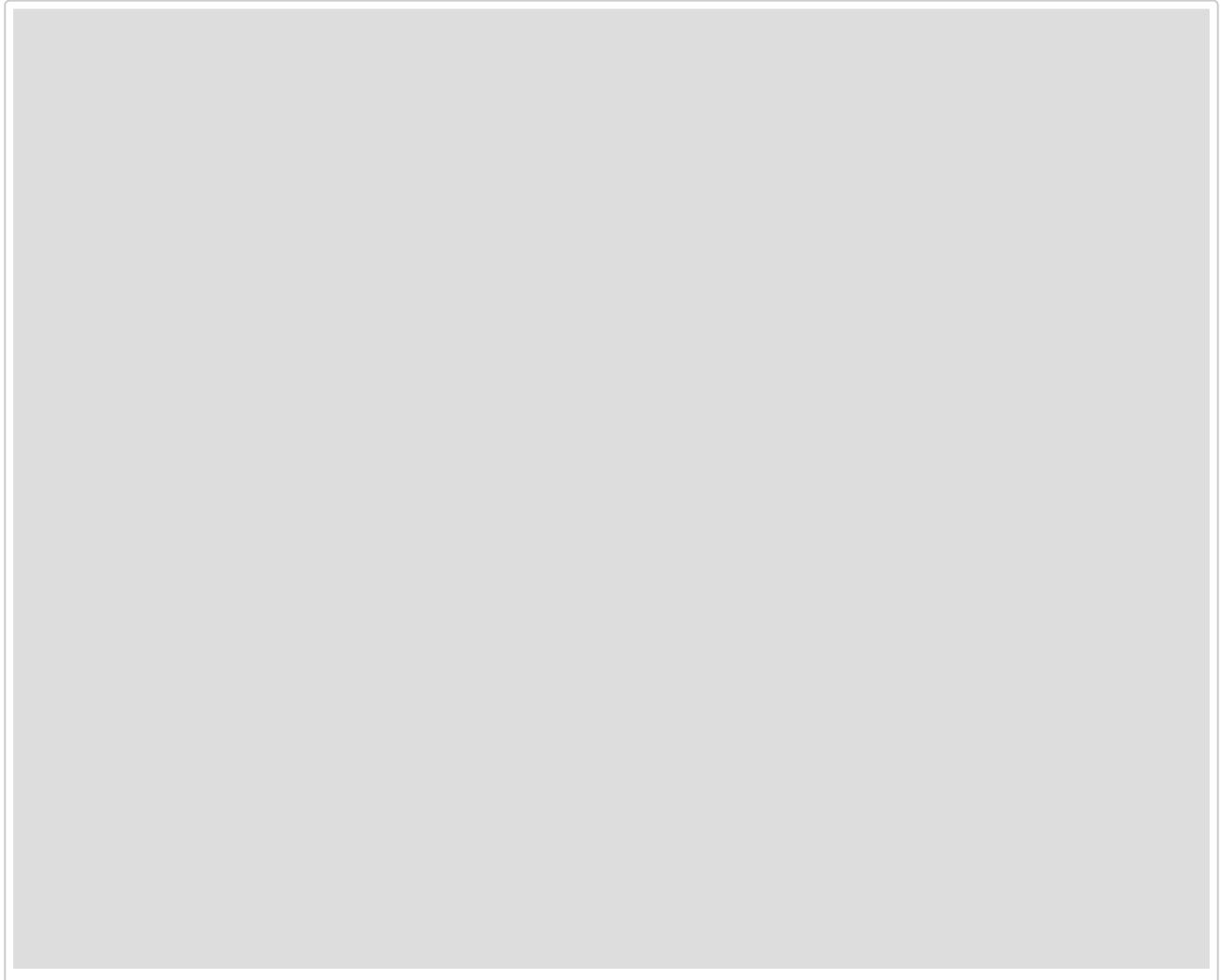
9.20 NGA - version 2.6



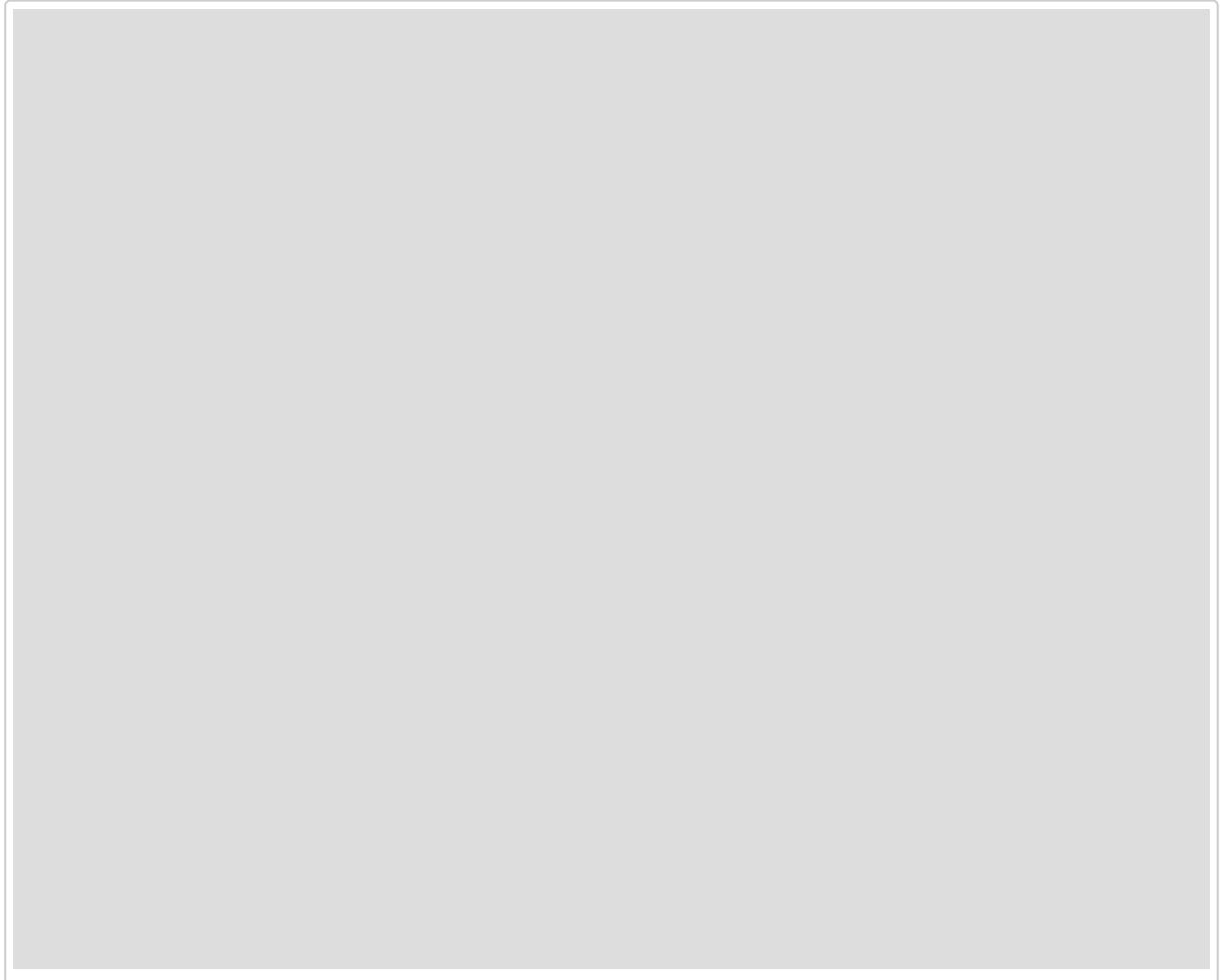
9.21 NGA - version 2.7



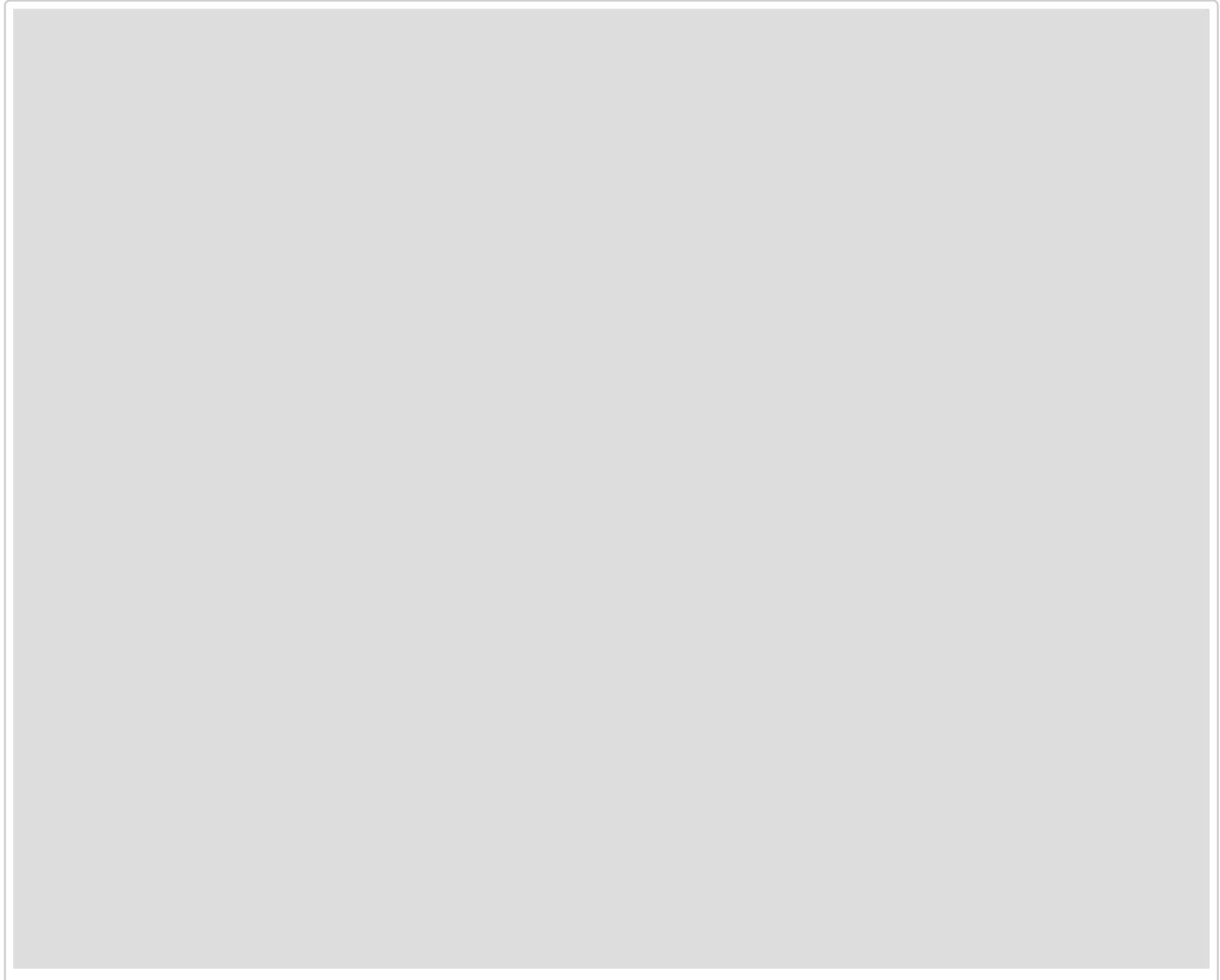
9.22 NGA - version 2.8



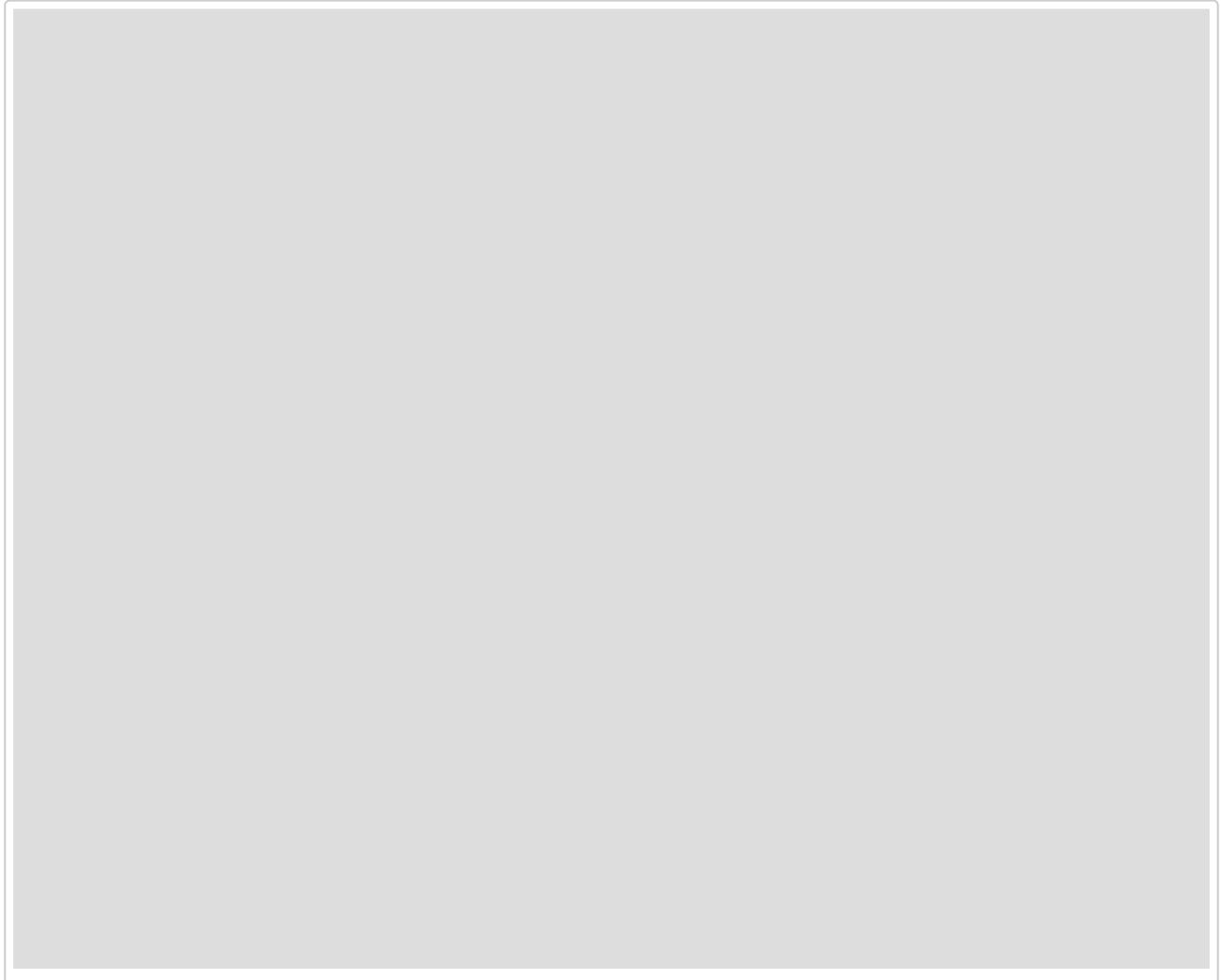
9.23 NGA - version 2.9



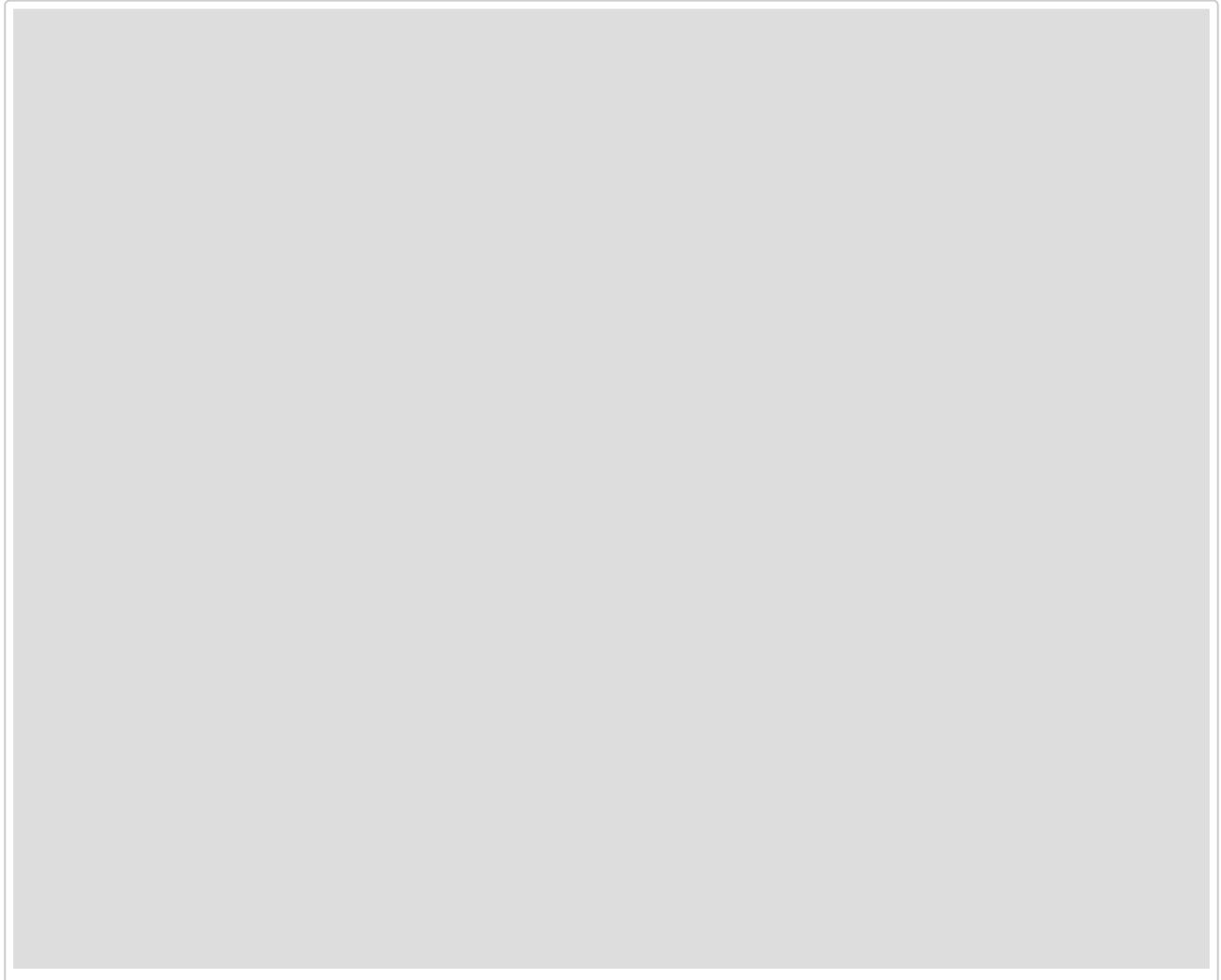
9.24 NGA - version 2.10



9.25 NGA - version 2.11



9.26 NGA - version 3.1



9.27 NGA - version 3.2

