Using Artificial Intelligence (AI)

For Large Software Engineering Projects – Part 2

Capers Jones

Foreword: The following pages constitute Part 2 of *Using Artificial Intelligence for Large Software Engineering Projects* by Capers Jones, released for distribution to the International Function Point Users Group (IFPUG) in January of 2025. Topics in this extract include: Best to Worst Software Engineering Practices.

Acknowledgment: The IFPUG community expresses its appreciation to Mr. Jones for his lifelong pursuit of metrics-based state of and recommended improvements for software development practices globally.

Joe Schofield

Software Engineering Progress from 1955 to 2025

While Table 2 showed that software engineering in 2025 is plagued by many serious problems, there have been significant improvements too. The author's team has developed a technology scoring method that ranks software technologies on a scale of +10 to -10.

A new method without much empirical data is the use of artificial intelligence (AI) to construct new applications. The combination of AI and suites of reusable components may possibly increase productivity from today's average of 8 function points per month to more than 800 function points per staff month.

AI can shorten development schedules for large systems in the 10,000-function point size range from 3 calendar years to 3 weeks, with 2 of those weeks devoted to discussions of user requirements before construction begins.

Although AI is not yet a widely deployed software engineering tool, it may have a larger impact on software productivity, costs, and schedules than any other in the history of software engineering.

The best current methods benefit quality and productivity by > 25% compared to average values. The worst methods at the bottom degrade productivity and quality by more than – 25% compared to average values. Table 3 shows the Namcook rankings for 275 software methods and practices:

	Table 3: Software Methodology Rankings	**Scores**
	Best Practices	
1	**Artificial intelligence (AI) used for large software systems**	**10.00**
2	**Reuse-oriented development (85% reusable materials)**	**10.00**
3	**Reuse certification to near zero-defect levels**	**10.00**
4	**Peak defect removal efficiency > 99%**	**9.67**
5	**Requirements patterns - InteGreat**	**9.50**
6	**Defect potentials < 3.00 per function point**	**9.35**
7	**Requirements modeling (T-VEC)**	**9.33**
8	**Average defect removal efficiency > 95%**	**9.32**
9	**Personal Software Process (PSP)**	**9.25**
10	**Team Software Process (TSP)**	**9.18**
11	**Automated static analysis - code**	**9.17**
12	**Mathematical test case design (Hexawise)**	**9.17**
13	**Formal Inspections (code)**	**9.15**
14	**Measurement of defect removal efficiency (DRE)**	**9.08**

15	**Hybrid (CMM+TSP/PSP+others)**	**9.06**
16	**Automated static analysis - text**	**9.00**
17	**Feature driven (FDD)**	**9.00**
18	**FOG readability index - requirements**	**9.00**
19	**Hybrid (agile/RUP/TSP)**	**9.00**
20	**IntegraNova**	**9.00**
21	**Kaizen/kanban**	**9.00**
22	**Model-driven development**	**9.00**
23	**Object Oriented (OO) development**	**9.00**
24	**Reusable feature certification**	**9.00**
25	**Reusable feature change controls**	**9.00**
26	**Reusable feature recall method**	**9.00**
27	**Reusable feature warranties**	**9.00**
28	**Reusable source code (zero defect)**	**9.00**
29	**SEMAT + TSP**	**9.00**
30	**TSP/PSP**	**9.00**
31	**T-VEC**	**9.00**

Good Practices

32	Activity-based productivity measures	8.83
33	Continuous integration	8.83
34	Early estimates of defect potentials	8.83
35	FLESCH readability score requirements	8.83
36	Object-oriented development (OO)	8.83
37	Automated security testing	8.58
38	Automated maintenance work benches	8.50
39	Automated UML static analysis	8.50
40	Measurement of bad-fix injections	8.50
41	Prince2 development methodology	8.50
42	Reusable test cases (zero defects)	8.50
43	Test case inspections	8.50
44	Formal security analysis	8.43
45	ITIL - customized	8.42
46	Formal Inspections (requirements)	8.40
47	Time boxing	8.38
48	Function point analysis (pattern matches)	8.33
49	Reusable designs (scalable)	8.33
50	Automated parametric cost estimating tools	8.28
51	Formal risk management	8.27
52	Automated defect tracking tools	8.17

53	Measurement of defect origins	8.17
54	Benchmarks against industry data	8.15
55	Function point analysis (high-speed)	8.15
56	Formal progress reports (weekly)	8.06
57	Agile development with SCRUM	8.02
58	Certification - vendor (Apple, Microsoft, etc.)	8.00
59	Certification (function points)	8.00
60	Continuous development	8.00
61	DevOps development methodology	8.00
62	Formal measurement programs	8.00
63	Global 24 hour development	8.00
64	Legacy data mining	8.00
65	Open-source	8.00
66	Project offices - automated	8.00
67	Prototypes - disposable	8.00
68	Re-estimating for requirements changes	8.00
69	Requirements modeling (IntegraNova)	8.00
70	Reusable architecture (scalable)	8.00
71	RUP	8.00
72	Service-Oriented modeling	8.00
73	Specifications by Example	8.00
74	Inspections (design)	7.94
75	Lean Six-Sigma	7.94
76	Six-sigma for software	7.94
77	Formal cost tracking reports	7.89
78	Automated configuration control	7.86
79	ITIL - normal	7.83
80	Formal test plans	7.81
81	Automated unit testing	7.75
82	Automated sizing tools (function points)	7.73
83	Automated project management tools	7.71
84	Scrum session (daily)	7.70
85	Automated quality and risk prediction	7.69
86	Reusable requirements (scalable)	7.67
87	Specialists for key skills	7.67
88	Formal requirements analysis	7.63
89	Data mining for business rule extraction	7.60
90	High-level languages (current)	7.53
91	Reusable tutorial materials	7.50
92	Rational Unified Process (RUP)	7.48
93	Function point analysis (IFPUG)	7.37

94	Measurement of requirements changes	7.37
95	Formal architecture for large applications	7.36
96	Automated ERP estimates	7.33
97	Best-practice analysis before start	7.33
98	Reusable feature catalog	7.33
99	Quality function deployment (QFD)	7.32
100	Joint Application Design (JAD)	7.27
101	TickIT assessments	7.25
102	Automated test coverage analysis	7.23
103	Measurement of defect severity levels	7.13
104	Formal SQA team	7.10
105	Inspections (test materials)	7.04
106	Automated project offices (APO)	7.00
107	DMAIC	7.00
108	Data State Development Method DSDM	7.00
109	Evolutionary Development (EVO)	7.00
110	Hybrid (agile+waterfall)	7.00
111	Lean	7.00
112	Legacy renovation	7.00
113	Mashup	7.00
114	Microsoft solutions	7.00
115	Product Line engineering	7.00
116	Reengineering	7.00
117	Reusable construction plans	7.00
118	Reusable HELP information	7.00
119	Reusable test scripts	7.00
120	Reverse engineering	7.00
121	SEMAT+Agile	7.00
122	Test-driven development	7.00
123	Total Cost of Ownership (TCO) measures	7.00

Average Practices

124	Formal Governance	6.92
125	Automated deployment support	6.87
126	Automated cyclomatic complexity analysis	6.83
127	Forensic analysis of cancelled projects	6.83
128	Reusable reference manuals	6.83
129	Capability Maturity Model (CMMI® Level 5)	6.82
130	Automated documentation tools	6.79
131	Annual training (technical staff)	6.67

132	Metrics conversion (automated)	6.67
133	Change review boards	6.62
134	Automated test case generation	6.50
135	Automated test library control	6.50
136	Formal scope management	6.50
137	Inspections (architecture)	6.50
138	Project offices - manual	6.50
139	Service level agreements (SLA)	6.50
140	Annual training (managers)	6.33
141	Extreme programming (XP)	6.28
142	Service-Oriented Architecture (SOA)	6.26
143	Automated requirements tracing	6.25
144	Automated performance analysis	6.17
145	Baselines for process improvement	6.17
146	Use cases	6.17
147	Information engineering (IE)	6.00
148	Iterative development	6.00
149	Legacy redevelopment	6.00
150	Spiral development	6.00
151	Structured development	6.00
152	User satisfaction surveys	6.00
153	Value analysis (intangible value)	6.00
154	Formal project office for large systems	5.88
155	Automated modeling/simulation	5.83
156	Certification (six sigma)	5.83
157	Outsourcing (maintenance => (CMMI® Level 3)	5.83
158	Capability Maturity Model (CMMI® Level 4)	5.79
159	Certification (software quality assurance)	5.67
160	Outsourcing (development => CMMI® Level 3)	5.67
161	Value analysis (tangible value)	5.67
162	Root-cause analysis	5.50
163	Total Cost of Learning (TOL) measures	5.50
164	Cost of quality (COQ)	5.42
165	Embedded users in team	5.33
166	Normal structured design	5.17
167	Capability Maturity Model (CMMI® Level 3)	5.06
168	Certification (project managers)	5.00
169	Certification (test personnel)	5.00
170	Crystal	5.00
171	Earned-value measures (EVA)	5.00
172	Merise	5.00

173	Prince 2	5.00
174	RICE objects	5.00
175	Unified Modeling Language (UML)	5.00
176	Normal maintenance activities	4.54
177	Rapid application development (RAD)	4.54
178	Function point analysis (FISMA)	4.50
179	Function point analysis (Netherlands)	4.50
180	Partial code reviews	4.42
181	Automated restructuring	4.33
182	Function point analysis (COSMIC)	4.33
183	Function point analysis (unadjusted)	4.33
184	Partial design reviews	4.33
185	Team Wiki communications	4.33
186	Function points (micro .001 to 10)	4.17
187	Automated daily progress reports	4.08
188	CASE	4.00
189	Maintenance tool suites	4.00
190	Outsourcing (maintenance < CMMI® Level 3)	4.00
191	User stories	3.83
192	Outsourcing (offshore => CMMI® Level 3)	3.67
193	Goal-question metrics	3.50
194	Refactoring	3.33
195	Manual document production	3.17
196	Capability Maturity Model (CMMI® Level 2)	3.00
197	RAD	3.00
198	Technical debt - metaphor	3.00

Unsafe Practices

199	Clean-room development	2.50
200	Formal design languages	2.50
201	ISO Quality standards	2.00
202	Pair programming	2.00
203	Project-level productivity measures	2.00
204	V-Model	2.00
205	Function point analysis (backfiring)	1.83
206	Use Case points	1.67
207	Normal customer support	1.50
208	Partial governance (low risk projects)	1.00
209	Waterfall	1.00
210	Low-level languages (current)	1.00

211	Object-oriented metrics	0.33
212	Manual testing	0.17
213	Outsourcing (development < CMMI® 3)	0.17
214	Story points	0.17
215	**Manual change control**	**-0.50**
216	**Manual test library control**	**-0.50**
217	**Reusability (average quality materials)**	**-0.67**
218	**Prototypes - evolutionary**	**-1.00**
219	**Capability Maturity Model (CMMI® Level 1)**	**-1.50**
220	**Informal progress tracking**	**-1.50**
221	**Outsourcing (offshore < CMM 3)**	**-1.67**
222	**End-user development**	**-2.00**
223	**Inadequate test library control**	**-2.00**
224	**Pair programming**	**-2.00**
225	**Generalists instead of specialists**	**-2.50**
226	**Inadequate measurement of productivity**	**-2.67**
227	**Manual cost estimating methods**	**-2.67**
228	**Cost per defect metrics**	**-2.83**
229	**Inadequate customer support**	**-2.83**

Worst Practices

230	**Friction between stakeholders and team**	**-3.50**
231	**Informal requirements gathering**	**-3.67**
232	**Lines of code metrics (logical LOC)**	**-4.00**
233	**Inadequate governance of financial applications**	**-4.17**
234	**Lines of code metrics (physical LOC)**	**-4.50**
235	**Partial productivity measures (coding only)**	**-4.50**
236	**Inadequate sizing of key deliverables (code, documents, tests etc.)**	**-4.67**
237	**High-level languages (obsolete)**	**-5.00**
238	**Technical debt – metrics**	**-5.00**
239	**Inadequate communications among team**	**-5.33**
240	**Inadequate change control**	**-5.42**
241	**Inadequate value analysis of application**	**-5.50**
242	**Cowboy programming methodology**	**-5.67**
243	**Friction/antagonism among team members**	**-6.00**
244	**Inadequate manual cost estimating methods**	**-6.04**
245	**Inadequate risk analysis before starting**	**-6.17**
246	**Low-level languages (obsolete)**	**-6.25**
247	**Government mandates (short lead times)**	**-6.33**
248	**Inadequate testing**	**-6.38**

249	**Friction/antagonism among management**	**-6.50**
250	**Inadequate communications with stakeholders**	**-6.50**
251	**Inadequate measurement of quality**	**-6.50**
252	**Inadequate problem reportings**	**-6.67**
253	**Error-prone modules (EPM) in applications**	**-6.83**
254	**Friction/antagonism among stakeholders**	**-6.83**
255	**Failure to estimate requirements changes**	**-6.85**
256	**Inadequate defect tracking methods**	**-7.17**
257	**Layoffs/loss of key personnel**	**-7.33**
258	**Rejection of estimates for business reasons**	**-7.33**
259	**Inadequate inspections of key deliverables**	**-7.42**
260	**Inadequate security controls**	**-7.48**
261	**Excessive schedule pressure by clients, management**	**-7.50**
262	**Inadequate progress tracking that hides problems**	**-7.50**
263	**Litigation (non-compete violation)**	**-7.50**
264	**Defective test cases with bugs of their own**	**-7.67**
265	**Inadequate cost tracking that omits unpaid overtime**	**-7.75**
266	**Litigation (breach of contract)**	**-8.00**
267	**Cowboy development methodology**	**-9.00**
268	**Defect potentials > 6.00 per function point**	**-9.00**
269	**Reusability (high defect volumes)**	**-9.17**
270	**Defect removal efficiency < 85%**	**-9.18**
271	**Litigation (poor quality/damages)**	**-9.50**
272	**Litigation (security flaw damages)**	**-9.50**
273	**Anti patterns or extremely poor development**	**-10.00**
274	**Litigation (intellectual property theft)**	**-10.00**
275	**Litigation (patent violation)**	**-10.00**

As can be seen by Table 3, the software engineering world has hundreds of methodologies and practices which range from very good to very harmful. Progress in software engineering resembles a "drunkard's walk" with a mix of advances and regressions. The reason for this is the endemic use of bad metrics such as "lines of code" (LOC) and "cost per defect" which conceal true progress and distort reality.

These poor measurement practices make selecting a software engineering methodology or selecting a programming language somewhat similar to joining a religious cult. In religious cults faith in invisible phenomena are substitutes for scientific deliberation based on actual empirical data.