

A Journey through Test Automation Patterns

One team's adventures with the Test Automation Patterns wiki

by Seretta Gamba and Dorothy Graham

Contents

Foreword.....	IX
Preface.....	X
1. Part 1.....	1
1.1. An (almost true) story	1
1.1.1. A chance meeting	1
1.1.2. Discovering the patterns and a way forward	7
1.1.3. Liz makes some changes.....	19
1.1.4. Abstraction Hamburgers and Keyword Furniture	27
1.1.5. Automation Trains and Object Map Desserts	36
1.1.6. Tim's accident.....	44
1.1.7. Testing Know-how	54
1.1.8. Jerry comes on-board.....	58
1.1.9. The Root of the Problems.....	64
1.1.10. Tom & Jerry	69
1.1.11. Finding the Best Solution.....	80
1.1.12. Not Only Standards.....	91
1.1.13. Frameworking.....	98
1.1.14. Implementing Solutions.....	110
1.1.15. Management Problems	118
1.1.16. New company, new Issues	125
2. Part 2.....	130
2.1. Mind Maps	130
2.1.1. General overview.....	131
2.1.2. Issues by category.....	133
2.1.3. Patterns by category.....	137
3. Part 3.....	140
3.1. Issues in Detail	140
3.1.1. <i>AD-HOC AUTOMATION</i> (Management Issue).....	140
3.1.2. <i>AUTOMATION DECAY</i> (Process Issue).....	141
3.1.3. <i>BRITTLE SCRIPTS</i> (Design Issue)	142
3.1.4. <i>BUGGY SCRIPTS</i> (Process Issue)	143
3.1.5. <i>CAN'T FIND WHAT I WANT</i> (Design Issue)	144
3.1.6. <i>COMPLEX ENVIRONMENT</i> (Design Issue)	145

3.1.7.	<i>DATA CREEP (Process Issue)</i>	146
3.1.8.	<i>DATE DEPENDENCY (Design Issue)</i>	147
3.1.9.	<i>FALSE FAIL (Execution Issue)</i>	148
3.1.10.	<i>FALSE PASS (Execution Issue)</i>	149
3.1.11.	<i>FLAKY TESTS (Execution Issue)</i>	150
3.1.12.	<i>GIANT SCRIPTS (Design Issue)</i>	151
3.1.13.	<i>HARD-TO-AUTOMATE (Design Issue)</i>	152
3.1.14.	<i>HARD-TO-AUTOMATE RESULTS (Design Issue)</i>	153
3.1.15.	<i>HIGH ROI EXPECTATIONS (Management Issue)</i>	154
3.1.16.	<i>INADEQUATE COMMUNICATION (Process Issue)</i>	156
3.1.17.	<i>INADEQUATE DOCUMENTATION (Process Issue).....</i>	157
3.1.18.	<i>INADEQUATE RESOURCES (Execution Issue).....</i>	157
3.1.19.	<i>INADEQUATE REVISION CONTROL (Process Issue).....</i>	158
3.1.20.	<i>INADEQUATE SUPPORT (Management Issue).....</i>	159
3.1.21.	<i>INADEQUATE TEAM (Management Issue)</i>	159
3.1.22.	<i>INADEQUATE TECHNICAL RESOURCES (Management Issue).....</i>	161
3.1.23.	<i>INADEQUATE TOOLS (Management Issue)</i>	161
3.1.24.	<i>INCONSISTENT DATA (Design Issue).....</i>	162
3.1.25.	<i>INEFFICIENT EXECUTION (Execution Issue)</i>	163
3.1.26.	<i>INEFFICIENT FAILURE ANALYSIS (Execution Issue)</i>	164
3.1.27.	<i>INADEQUATE TEAM (Management Issue)</i>	166
3.1.28.	<i>INFLEXIBLE AUTOMATION (Execution Issue).....</i>	167
3.1.29.	<i>INTERDEPENDENT TEST CASES (Design Issue)</i>	168
3.1.30.	<i>INSUFFICIENT METRICS (Process Issue).....</i>	168
3.1.31.	<i>KNOW-HOW LEAKAGE (Management Issue)</i>	169
3.1.32.	<i>LATE TEST CASE DESIGN (Process Issue)</i>	170
3.1.33.	<i>LIMITED EXPERIENCE (Management Issue)</i>	171
3.1.34.	<i>LITTER BUG (Execution Issue).....</i>	175
3.1.35.	<i>LOCALISED REGIMES (Management Issue)</i>	175
3.1.36.	<i>LONG SET-UP (Design Issue).....</i>	176
3.1.37.	<i>MANUAL INTERVENTIONS (Execution Issue).....</i>	177
3.1.38.	<i>MANUAL MIMICRY (Design Issue).....</i>	178
3.1.39.	<i>MULTIPLE PLATFORMS (Design Issue)</i>	180
3.1.40.	<i>NO INFO ON CHANGES (Process Issue)</i>	181
3.1.41.	<i>NO PREVIOUS TEST AUTOMATION (Management issue)</i>	182
3.1.42.	<i>NON-TECHNICAL-TESTERS (Process Issue)</i>	183

3.1.43. <i>OBSCURE MANAGEMENT REPORTS (Management Issue)</i>	184
3.1.44. <i>OBSCURE TESTS (Design Issue)</i>	184
3.1.45. <i>REPETITIOUS TESTS (Design Issue)</i>	186
3.1.46. <i>SCHEDULE SLIP (Management Issue)</i>	186
3.1.47. <i>SCRIPT CREEP (Process issue)</i>	187
3.1.48. <i>STALLED AUTOMATION (Process Issue)</i>	188
3.1.49. <i>SUT REMAKE (Management Issue)</i>	190
3.1.50. <i>TEST DATA LOSS (Process Issue)</i>	191
3.1.51. <i>TOO EARLY AUTOMATION (Design Issue)</i>	192
3.1.52. <i>TOOL DEPENDENCY (Design Issue)</i>	193
3.1.53. <i>TOOL-DRIVEN AUTOMATION (Process Issue)</i>	194
3.1.54. <i>UNAUTOMATABLE TEST CASES (Design Issue)</i>	195
3.1.55. <i>UNFOCUSSED AUTOMATION (Process Issue)</i>	196
3.1.56. <i>UNMOTIVATED TEAM (Management Issue)</i>	197
3.1.57. <i>UNREALISTIC EXPECTATIONS (Management issue)</i>	198
4. Part 4	200
4.1. Patterns in Detail	200
4.1.1. <i>ABSTRACTION LEVELS (Design Pattern)</i>	200
4.1.2. <i>ASK FOR HELP (Process Pattern)</i>	202
4.1.3. <i>AUTOMATE EARLY (Management Pattern)</i>	203
4.1.4. <i>AUTOMATE GOOD TESTS (Design Pattern)</i>	203
4.1.5. <i>AUTOMATE THE METRICS (Design Pattern)</i>	205
4.1.6. <i>AUTOMATE WHAT'S NEEDED (Process Pattern)</i>	206
4.1.7. <i>AUTOMATION ROLES (Management Pattern)</i>	207
4.1.8. <i>CAPTURE-REPLAY (Design Pattern)</i>	209
4.1.9. <i>CELEBRATE SUCCESS (Process Pattern)</i>	210
4.1.10. <i>CHAINED TESTS (Design Pattern)</i>	211
4.1.11. <i>CHECK-TO-LEARN (Process Pattern)</i>	212
4.1.12. <i>COMPARE WITH PREVIOUS VERSION (Execution Pattern)</i>	213
4.1.13. <i>COMPARISON DESIGN (Design Pattern)</i>	214
4.1.14. <i>DATA-DRIVEN TESTING (Design Pattern)</i>	215
4.1.15. <i>DATE INDEPENDENCE (Design Pattern)</i>	216
4.1.16. <i>DEDICATED RESOURCES (Management Pattern)</i>	217
4.1.17. <i>DEFAULT DATA (Design Pattern)</i>	218
4.1.18. <i>DEPUTY (Management Pattern)</i>	219
4.1.19. <i>DESIGN FOR REUSE (Design Pattern)</i>	220

4.1.20.	DO A PILOT (Management Pattern).....	222
4.1.21.	DOCUMENT THE TESTWARE (Process Pattern)	224
4.1.22.	DOMAIN-DRIVEN TESTING (Design Pattern).....	226
4.1.23.	DON'T REINVENT THE WHEEL (Design Pattern).....	227
4.1.24.	EASY TO DEBUG FAILURES (Execution Pattern)	228
4.1.25.	EXPECT INCIDENTS (Execution Pattern)	229
4.1.26.	EXPECTED FAIL STATUS (Execution Pattern).....	230
4.1.27.	FAIL GRACEFULLY (Execution Pattern).....	231
4.1.28.	FRESH SETUP (Design Pattern).....	232
4.1.29.	FULL TIME JOB (Process Pattern).....	234
4.1.30.	GET ON THE CLOUD (Management Pattern)	235
4.1.31.	GET TRAINING (Management Pattern)	236
4.1.32.	GOOD DEVELOPMENT PROCESS (Process Pattern)	237
4.1.33.	GOOD PROGRAMMING PRACTICES (Process Pattern).....	239
4.1.34.	INDEPENDENT TEST CASES (Design Pattern)	240
4.1.35.	KEEP IT SIMPLE (Design Pattern).....	242
4.1.36.	KEYWORD-DRIVEN TESTING (Design Pattern)	243
4.1.37.	KILL THE ZOMBIES (Process Pattern)	245
4.1.38.	KNOW WHEN TO STOP (Management Pattern)	245
4.1.39.	LAZY AUTOMATOR (Process Pattern)	247
4.1.40.	LEARN FROM MISTAKES (Process Pattern)	248
4.1.41.	LOOK AHEAD (Process Pattern)	250
4.1.42.	LOOK FOR TROUBLE (Process Pattern)	250
4.1.43.	MAINTAIN THE TESTWARE (Process Pattern).....	252
4.1.44.	MAINTAINABLE TESTWARE (Design Pattern).....	253
4.1.45.	MANAGEMENT SUPPORT (Management Pattern)	255
4.1.46.	MIX APPROACHES (Management Pattern)	257
4.1.47.	MODEL-BASED TESTING (Design Pattern).....	258
4.1.48.	OBJECT MAP (Execution Pattern).....	259
4.1.49.	ONE CLEAR PURPOSE (Design Pattern).....	260
4.1.50.	ONE-CLICK RETEST (Execution Pattern)	261
4.1.51.	PAIR UP (Process Pattern).....	262
4.1.52.	PARALLELIZE TESTS (Execution Pattern)	263
4.1.53.	PLAN SUPPORT ACTIVITIES (Management Pattern)	264
4.1.54.	PREFER FAMILIAR SOLUTIONS (Management Pattern)	266
4.1.55.	PRIORITIZE TESTS (Execution Pattern)	266

4.1.56.	READABLE REPORTS (Design Pattern)	268
4.1.57.	REFACTOR THE TESTWARE (Process Pattern)	269
4.1.58.	RIGHT INTERACTION LEVEL (Design Pattern)	270
4.1.59.	RIGHT TOOLS (Management Pattern)	271
4.1.60.	SELL THE BENEFITS (Management Pattern)	274
4.1.61.	SENSITIVE COMPARE (Design Pattern)	275
4.1.62.	SET CLEAR GOALS (Management Pattern)	277
4.1.63.	SET STANDARDS (Process Pattern)	279
4.1.64.	SHARE INFORMATION (Process Pattern)	282
4.1.65.	SHARED SETUP (Design Pattern)	284
4.1.66.	SHORT ITERATIONS (Process Pattern)	285
4.1.67.	SIDE-BY-SIDE (Management Pattern)	286
4.1.68.	SINGLE PAGE SCRIPTS (Design Pattern)	287
4.1.69.	SKIP VOID INPUTS (Execution Pattern)	288
4.1.70.	SPECIFIC COMPARE (Design Pattern)	288
4.1.71.	STEEL THREAD (Execution Pattern)	289
4.1.72.	TAKE SMALL STEPS (Process Pattern)	290
4.1.73.	TEMPLATE TEST (Design Pattern)	291
4.1.74.	TEST AUTOMATION BUSINESS CASE (Management Pattern)	292
4.1.75.	TEST AUTOMATION FRAMEWORK (Design Pattern)	295
4.1.76.	TEST AUTOMATION OWNER (Management Pattern)	296
4.1.77.	TEST SELECTOR (Design Pattern)	297
4.1.78.	TEST THE TESTS (Process Pattern)	299
4.1.79.	TESTABLE SOFTWARE (Management Pattern)	300
4.1.80.	TESTWARE ARCHITECTURE (Design Pattern)	302
4.1.81.	THINK OUT-OF-THE-BOX (Design Pattern)	304
4.1.82.	TOOL INDEPENDENCE (Design Pattern)	305
4.1.83.	UNATTENDED TEST EXECUTION (Execution Pattern)	306
4.1.84.	VARIABLE DELAYS (Execution Pattern)	307
4.1.85.	VERIFY-ACT-VERIFY (Design Pattern)	308
4.1.86.	VISUALIZE EXECUTION (Execution Pattern)	310
4.1.87.	WHOLE TEAM APPROACH (Process Pattern)	310
5.	Appendix	312
5.1.	Recipes	312
5.1.1.	Tiramisu	312
5.1.2.	Lentil soup	312

5.1.3.	Eggplant Parmigiana	313
5.1.4.	Bolognese Sauce	314
5.1.5.	Liz's special salad.....	314
5.1.6.	Toffee sauce (topping for ice cream)	315
5.1.7.	Chicken Hungarian style.....	315
5.2.	Overview of relationships between story characters.....	319
5.3.	Index.....	320

Foreword

Is being written by Isabel Evans

Preface

This book

This is a story - which could be real - about a software test automator and her struggles to implement good automation in her company.

Our aim in this book is to show you how you can use the test automation patterns wiki to help with various test automation problems, by showing how our heroine, Liz, copes with many of them.

This book started as a way to explain how the patterns in the wiki TestAutomationPatterns.org could be used in realistic situations, warts and all. Things are never as straight-forward as we hope, and this book shows a number of things that go wrong, as well as many things that go well.

In this book you will encounter the Hamburger model of test automation abstraction, as well as the Dessert model of object selection. You will get to know Liz, her hobbies, her friends, her unrequited love... In short, yes, this is a story but we hope it will help make the patterns "come alive"!

Where did these patterns come from?

In 2012, the book Experiences in Test Automation, by Dorothy Graham and Mark Fewster was published by Addison Wesley (ISBN 978-0-321-75406-6). This book collected case studies of test automation from a number of different contributors, including Seretta Gamba (Chapter 21, Automation Through the Back Door (by Supporting Manual Testing)).

When Seretta got her copy of the complete book and read through all the other stories, she thought, "These are patterns!" As a developer, she was familiar with development patterns, but recognised that many of the case studies followed similar ways of doing things in automation that led to success. She then began writing up the patterns, and asked Dorothy to review and participate in improving them.

We soon realised that text documents didn't work because the patterns and issues are so interconnected, so we made a wiki. From 2013 until 2018, the wiki was hosted on wikispaces.com, but when they closed down in 2018, we were delighted that Test Huddle was willing to take it over.

How to read this book

You can read this book just as a story about Liz and her life - if so, we hope you are entertained by her experiences.

Or you can read this book and take note of the patterns that are discussed by Liz and her co-workers. We include the relevant parts of the patterns in the text so that you can read the book without having to look up the pattern in the wiki (for example if you are reading it without internet access), but you could also look up the patterns in the wiki to see the full text if you want.

We hope that, inspired by Liz, you will want to apply the patterns to some of your own test automation issues and problems!

Each chapter starts with a Haiku, a Japanese poem of 17 syllables, 5, 7 and 5 in the 1st, 2nd and 3rd lines respectively. We liked this idea from Isabel Evans and took it as a challenge to try to summarise something about each chapter in this short verse.

After the story, we include some mind-map overviews of issues and patterns, and we also list all of the issues and all of the patterns in alphabetical order. This is so that you can see the full content even without having online access to the wiki.

Feedback about this book and/or the wiki

We are happy to hear from anyone who has feedback for us either about this book or about the wiki: experiences to relate, suggestions to improve the wiki, or new issues or patterns that would be helpful to others. Please use the “Feedback” page in the wiki.

Disclaimer

All names of companies or people are invented. Any resemblance to actual people or companies is purely coincidental.

1. Part 1

1.1. An (almost true) story

1.1.1. A chance meeting

Nobody helps.
Test Automation is stuck.
Coffee with a friend!

“It was such a dream job, how could I imagine that they would be so....stupid!” exclaimed Liz angrily. In front of her sat Tigg, the lead singer of her jazz band. Leaned against the near wall were their instruments, Liz’s bass and Tigg’s saxophone. They were sitting in the café where they regularly went after a Saturday practice session. This time the other band members hadn’t had time and so Liz and Tigg were sitting alone.

“Now calm down and start again from the beginning,” said Tigg. “When you’re angry you talk so fast that I only understand one word in three!”

“I’m sorry,” answered Liz, “I really am angry. You remember me telling you what an awesome job I had finally found: test automator at Digiphonia, the very company that developed J(azz)-Screech, the app that helped us compose our last two songs?”

“Yes,” answered Tigg, “ever since we got the Screech-App you’ve always been talking about it, and you were really excited about starting to work there and getting the chance to restructure their stalled automation!” She went on to say, “But it hasn’t been that long, what’s happened to make you so angry?”

“Well it all started really well,” said Liz. “After doing all the ‘starting at a new company’ stuff - badges, desk, computer, meet the new colleagues etc., I was given all kinds of informative material about the apps, how they started, how they work, what they are planning to do next and so on... That was really awesome!”

“Can you believe how the company began?” continued Liz. “Tom, the managing director, had had the idea for a composition app as a present for his Dad Paul’s 60th birthday. His Dad plays in a rock group the D-eatles, and they play Beatles music (with expensive royalties) but also new songs that sound like the Beatles. Paul liked to write new lyrics, but had trouble composing the music. Tom had a college friend who was not only an excellent mathematician, but also a real computer geek and he asked him if he could write an app to compose music. And imagine, his name is Jerry!” she added laughing “The bosses of Digiphonia are Tom & Jerry!”

Tigg stared “You’re not kidding?” she asked. “Really? Tom & Jerry? And are they really cat and mouse?”

Liz laughed “Actually I haven’t found out yet. From what I’ve seen, Tom is the typical manager, all numbers, ROI and shareholder value. Jerry, I haven’t even met yet. Anyway, apparently, he was immediately taken by the idea and soon came up

with B(eatle)-Screech. Paul loved his birthday present, and so did everyone else at the party. Soon Tom and Jerry were asked to do other apps in other styles, even classical music: Ba(ch)-Screech and Moz(art)-Screech. As the apps took off, Tom and Jerry realised they could make a living out of this, so they left their former jobs and founded Digiphonia. I learned all this from Paul. He has become in effect the principal tester because he doesn't trust normal testers to notice the difference between what he calls a 'screech' and real music. He was delighted when he found out that I play in a band and we had a long discussion about J-Screech and how to improve it."

"But that just sounds great," Tigg said "Your dream job as you thought. So why are you so angry?"

"Well, then I should have been introduced to the technical aspects of the apps by Jerry, but he was away at some conference. So, I wanted to ask the testers about their testing and test automation, but I was told not to approach them until after the weekly release as they would be far too busy with testing."

"I see," said Tigg "you aren't getting anywhere, right? Knowing you, that must be a real punishment," she added with a grin.

Suddenly Liz said, "Why, if that isn't Jack!" She stood up and called out, "Hey, Jack, come over here and sit with us!" and she was already running to a friendly looking guy with a coffee cup in his hands looking for a place to sit.

"Hi, Liz" exclaimed Jack "haven't seen you for ages. What are you up to?"

Liz led him to their table and asked him "Do you recognize Tigg with her new braids?" And to Tigg she added "And do you remember Jack, my test automation mentor at InsureSafe?"

Jack smiled at Tigg and answered "Hello, beautiful! Just a few hundred braids can't change anybody so much! You look good, but I also liked your Afro-look!" and then added "but I see you both sit at a small table, I'm waiting for Deirdre."

"Oh, I'm leaving anyhow" said Tigg standing up "I was expecting my Mom and I see her just coming in."

In the direction she was looking at, they saw a plump little woman holding a big plastic container clearly searching for somebody. Tigg waved her over and as her mother approached she introduced her to them "Liz, Jack, this is my Mom" and turning to her mother she added "Mom, this is Liz and here is Jack, an old colleague of hers."

With a wide smile Tigg's mother said to Liz "Hi, how nice to finally meet you! Tigerlily is always talking about you. I've brought some cookies, try one."

"Tigerlily?" mouthed Liz to Tigg, who grimaced and, interrupting her mother, said "Let's go, Mom, Liz has to discuss some office stuff with Jack and...PLEASE don't call me Tigerlily! Liz, you'll have to fill me in on the rest of the story after our next practice session."

Still glowering, she picked up her saxophone and, gently nudging her mother toward the door, she waved good bye to Liz and Jack.

"I didn't want to break you up" muttered Jack, sitting down. "What's going on?" Liz was really happy to see Jack. Talking to Tigg had helped, but of course she could not really 'talk shop' with her. After the usual 'casual encounter small-talk' she repeated what she had just told Tigg about how well it had started, and how she couldn't get any information from the testers. Jack wanted to know what tool they were using, so Liz replied, "To start with they bought AutomatePro. As you know, this is a state-of-the-art commercial test automation tool, and Tom and Jerry thought that this would solve all their regression testing problems."

"Oh-Oh," chuckled Jack "I think I know what happened next. Let me guess, the tool has a feature to develop keywords just by running the application. In the blink of an eye you can develop all kinds of keyword tests, so even non-technical testers can immediately get going, right?"

"Awesome," confirmed Liz "That's exactly what happened. One of the testers, Ronald, who was quite keen on using the new tool, got started right away and was able to automate lots of tests very quickly. However, what they didn't realize was that this feature was more or less a 'glorified' Capture-Replay that seduces people to create all kinds of keywords so fast that they don't notice that they are repeating the same stuff over and over again. Everybody was happy to have so much automation so fast. The rude awakening came with the next release, when they had to update practically all their keywords! Development of automated tests came to a screeching halt; now they only barely manage to keep the old ones running."

"Well, "a screeching halt" sounds appropriate for Digiphonia," Laughed Jack. "But this sounds like a perfect job for you, Liz. Remember when I had the same problem with our automation at InsureSafe, and you helped me to structure the automation code so that it was maintainable with far less effort. I guess you would need to get to grips with the tool, but having used other tools, I'm sure you can cope with that. This doesn't sound like an insurmountable challenge!"

"You're right," replied Liz "the technical stuff is not the problem here, and I was actually looking forward to getting my hands dirty with the automation code. No, the problem is with the people!"

"Ah, 'there's the rub,'" said Jack. "There's a common perception that automation is just about tools and technical stuff, but it's definitely not!"

"You're really good with people, Jack. I'm hoping you can give me some advice about what to do next, I feel like I could slap each and every one there."

"Now, now," Jack smiled at her, "Take it easy! Tell me what happened, and I'll see if I can think of anything to help."

Liz continued with the experiences of her first week. "Since I was not allowed to approach the testers, and Jerry the development manager was away, I decided to

go and talk to the developers, so I went to the lead developer, Leroy, with all kinds of questions. But instead of getting the technical information I needed, I was confronted by a guy who seemed to be quite angry. I remember the scene exactly..."

And she added with Leroy's voice and drawl "I don't have time to answer your stupid questions. I have more important things to do than waste my time for some absolutely useless test automation."

"I was quite surprised at this and replied: 'Excuse me? But they just hired me to do test automation! Why would it be useless?'"

Going on with Leroy's voice she continued "Sorry, I know that it's not your fault, but we urgently need a new developer and instead they hire a test automator! We have written unit tests for absolutely everything! What more do we need? Our tests run with every build every single day, sometimes even more than once a day. That's why it didn't matter that our testers made such a mess of system test automation."

At this point Jack's chuckle was turning into an outright laugh.

Liz continued, "To that I answered: 'Well, unit tests are fine, and I think it's great that you have so many, but system level tests should be executed as well, so it's important to automate as many as ...' - and here I was interrupted by Leroy."

And she concluded, again with Leroy's voice, "Oh, don't start with this stuff on me. I know that our application is really good, our customers are happy, what more do you want? Please go away and let me work."

Jack was now laughing so hard that he already had tears in his eyes. The people at the neighbouring tables were looking over curiously.

"You are not supposed to laugh at this story! It's not a joke!" exclaimed Liz trying to keep from laughing herself.

"You know, Liz" he said after he had finally regained his breath "If you ever want to change career, I suggest cabaret!"

"Ok," prompted Liz with a smile, "but now be serious."

Looking especially serious Jack added "It's good that the developers are doing automated unit tests, but it's surprising that there is so much antagonism towards system-level automation, or even system-level testing! I wonder if they all think this way."

"Well, I decided not to approach the other developers just yet."

"Very sensible - what happened next?"

Liz went on "Having nothing to do, I spent some time -days actually- exploring the tool. After the release, I finally went to see the testers. I expected, and hoped for, a very different welcome; after all I had been told that they were the ones who had requested that an experienced test automator be hired to help them. So, I went to one of the testers, Jill, and wanted to ask her all kinds of questions about the tests

for the apps, about what they had already automated, about the tool etc., but Jill seemed almost as unfriendly as Leroy was.”

And exaggerating Jill’s voice she continued “I’m sorry, but I really don’t have time right now. The next release is due soon and we have loads of testing to do before then.”

“As you can imagine I was again quite disappointed but tried not to show it. I had thought that after the release, the testers would have time for me, so I asked if she could just tell me where I could go to get the information I needed, but Jill was obviously trying to cut short the conversation.”

Continuing again with Jill’s voice Liz added “Look, why don’t you try to learn to use our tool AutomatePro. You are the expert automator after all. I’m a tester and I like it that way, I don’t intend to become a programmer! But as soon as we are finished with this next release, I’m sure we will have time to talk about the automation.”

Speaking now with her own voice Liz said “I was discouraged, but not ready to give up so I tried to get information from the other testers, Tim and Ronald. Same story! Or even worse, because it seems that at Digiphonia ‘after the release is before the release’! I realised then that this would be exactly what they would tell me next time as well. Tim explained to me that even though the testers were the ones who had been keen to get a test automator, the pressure of their other work was just piling on top of them. They were sorry, but they just didn’t see how they could spare any time to help me”.

Liz went on “Well, I had been hired by Tom to do test automation, so I needed to see what he would do about this – I was convinced that he would have to find a way to give the testers time to support me getting started! I tried to get to see either Tom or Jerry but of course they were both really busy too! Finally, late yesterday afternoon, I got to talk to Tom and explained the problem. Tom listened for a while, and then told me...”

Liz again changed her voice and accent and was now Tom. “Liz, I really expected more independence from such an experienced test automator. Just do your job, automate the missing test cases and let everyone else get on with their work. I know that test automation is valuable, and I support you 100%, but the daily work is much more important, that’s where our bread and butter are coming from.”

“So, Jack,” she concluded, again being herself “now you see why I am so angry. How can I do anything when no one will talk to me?”

“I would probably be angry too,” said Jack, “and I am not impressed with Tom saying he supports you ‘100%’ but then not giving you the time, you need from everyone. That’s not what I understand as support, that’s ‘lip service’”.

“So, what can I do? I’m not going to give up that easily!”

Just then Deirdre arrived with several bags of shopping. “Oh, I see that you’ve been very successful,” said Jack with a smile.

Liz and Deirdre hugged, and Liz observed a light bulge in Deirdre's belly, and saw some baby things in her shopping bag.

"Oh" she guessed "are two boys not enough yet?"

Both Deirdre and Jack laughed "Actually we are hoping for a girl this time," admitted Jack "but a boy would also be welcome."

After Deirdre heard about Liz's infuriating week, she asked, "Why don't you come for lunch tomorrow? The boys are with Jack's mother for the week-end, she's always happy to have them - it sounds like there is a bit more for the two of you to discuss."

"Wonderful! Thanks so much - I really appreciate it," said Liz.

When they all left, Jack helped Liz carry her bass. "Where's your car?" he asked.

"No car, my bicycle!" replied Liz with a grin, pointing to a well-used mountain bike tied to a small tree on the sidewalk, and after strapping the bass to her back she jumped on and pedalled off merrily whistling the tune of the piece they had practiced that morning.

Turning to his wife Jack mused "Well that's about as tough as you can get!"

1.1.2. Discovering the patterns and a way forward

Patterns will help solve
test automation issues.
Show me how to start!

Liz had thought of taking a bottle of wine to the lunch at Jack and Deirdre's, however she opted to make tiramisu¹ instead with a non-alcoholic portion. She hoped Deirdre wouldn't mind that she and Jack would be "talking shop" for a while at least.

When she arrived, Deirdre was happily busy in the kitchen (she loved to cook), so she sent Liz and Jack off. Jack brought Liz to his desk and pointed at the open notebook. "I've been thinking about your problem and I think I have just what you need." He added, "I was recently sent to a tutorial about Test Automation Patterns. Have you heard about them?" he asked.

Liz looked puzzled and reflected "I know there are some patterns for unit tests, but how can they help me?"

Jack corrected her "No, I'm not talking about unit tests. For a couple of years now, there has been a wiki with a collection of system level test automation patterns."

Liz looked up interested "And what kind of patterns are there? How are they different from the unit test patterns?"

"There are patterns for management, process, design and execution," Jack said, "and of course the issues that can be solved applying them. They are different from the unit test patterns because, first of all they are not prescriptive."

"What do you mean?" interrupted Liz.

"Well, unit-test or software design patterns take a development problem and give you exactly the code to solve it. You can implement it just as is and at the most you have to translate it into your own development language." And he continued, "Another difference is that the system-level patterns are more generalized, you can even say vague. Think for instance of management issues: companies can be so different in structure, size, hierarchy, whatever, that it would be impossible to give a single solution for every situation. That's why many of the patterns just suggest what kind of solutions one should try out."

"Interesting," said Liz. "I hadn't heard about them before. And how many are there?"

"Hmm, I think that right now there are some 70-80 patterns and about 50-60 issues."

"Oh my!" interrupted Liz again "How can you find the patterns that you need? Do you have to examine them one by one?"

¹ See appendix for recipes

"No, they've thought about that too!" exclaimed Jack with a grin. "There is a diagnostic functionality that lets you easily find the issue or issues that are bothering you and then the issue suggests what patterns to apply."

"Ah" answered Liz "that sounds awesome, could we try it out together?"

"That's why we're here!" said Jack still grinning and showed her a page on the screen. "I've skipped the very first diagnostic question because it was clear to me that what you want to do is to improve this automation."²

First question (when you want to improve or revive test automation)

What below describes the most pressing problem you have to tackle at the moment?

1. Lack of support (from management, testers, developers etc.).
2. Lack of resources (staff, software, hardware, time etc.).
3. Lack of direction (what to automate, which automation architecture to implement etc.).
4. Lack of specific knowledge (how to test the Software Under Test (SUT), use a tool, write maintainable automation etc.).
5. Management expectations for automation not met (Return On Investment (ROI), behind schedule, etc.).
6. Expectations for automated test execution not met (scripts unreliable or too slow, tests cannot run unattended, etc.).
7. Maintenance expectations not met (undocumented data or scripts, no revision control, etc.).

Liz considered the possible answers. "You know" she said "Actually any one of the answers would be conceivable! What do you do in such a case? How do you pick the right one?"

Jack smiled, "They taught us that the best strategy is to start with the topic that hurts the most. What do you think it is?"

Liz mulled Jack's answer over for a time and then decided "I would try for number 2 because in order to be able to work efficiently I need time and resources to develop the framework."

Jack clicked on the link, and it brought up a second question:

Second question (for "Lack of resources")

Please select what you think is the main reason for the lack of resources:

² A note from this book's authors: We will be showing the content of various pages from the Test Automation Patterns wiki, but we don't always show all of what is available online, just the aspects that are relevant to our story. Do have a look at the wiki pages yourself alongside reading about our characters!

For the following items look up the issue [AD-HOC AUTOMATION](#):

1. Expenses for test automation resources have not been budgeted.
2. There are not enough machines on which to run automation.
3. The databases for automation have to be shared with development or testing.
4. There are not enough tool licences.

More reasons for lack of resources:

1. No time has been planned for automation or it is not sufficient. Look up the issue [SCHEDULE SLIP](#) for help in this case.
2. Training for automation has not been planned for. See the issue [LIMITED EXPERIENCE](#) for good tips.
3. Nobody has been assigned to do automation, it gets done "on the side" when one has time to spare. The issue [INADEQUATE TEAM](#) should give you useful suggestions on how to solve this problem.

Liz was not really happy with the choices offered here “Hmm, perhaps number 1 under ‘More reasons’ could fit, but I’m not really sure.”

“Well, let’s see where that would bring us. If it still doesn’t fit, we can always go back. Let’s see first what the issue [SCHEDULE SLIP](#) suggests.”

SCHEDULE SLIP (Management Issue)

Examples:

1. Test automation is done only in people's spare time.
2. Team members are working on concurrent tasks which take priority over automation tasks.
3. Schedules for what automation can be done were too optimistic.
4. Necessary software or hardware is not available on time or has to be shared with other projects.
5. The planned schedule was not realistic.

“No, this really is not very helpful. Maybe this wiki isn’t such a good idea after all,” Liz scowled. “I was thinking more about time and support from my colleagues.”

“Then let’s go back to the first question and select another answer,” Jack suggested.

This time, Liz said, “Of course, number 1 is much more what I am struggling with, ‘Lack of support: from management, testers, developers, etc.’ Why did I skip this before?”

Jack clicked on this link and got the second question for this thread:

Second question (for “Lack of support”)

What kind of support are you missing?

If you are lacking support in a specific way, one of the following may give you ideas:

1. Managers say that they support you, but back off when you need their support. Probably managers don't see the value of test automation and thus give it a lower priority than for instance going to market sooner.
2. Testers don't help the automation team.
3. Developers don't help the automation team.
4. Specialists don't help the automation team with special automation problems (Databases, networks etc.).
5. Nobody helps new automators.
6. Management expected a "silver bullet" or magic: managers think that after they bought an expensive tool, they don't need to invest in anything else. See the issue UNREALISTIC EXPECTATIONS.

If you are having general problems with lack of support in many areas, the issue INADEQUATE SUPPORT may help.

"Awesome" said Liz, "here all the answers except number 4 could apply. How do we select the best one?"

"Select the one that is most pressing now."

Liz thought it over "Then I guess that would be answer number 2, 'Testers don't help the automation team' - with the 'team' just being me!"

"OK, let's see where that leads us," said Jack.

Third question (for "Testers don't help the automation team")

Please select what you think is the main reason for the lack of support from testers:

1. Testers think that the Software Under Test (SUT) is so complex that it's impossible to automate, so why try.
2. Testers don't have time because they have an enormous load of manual tests to execute.
3. Testers think that the SUT is still too unstable to automate and so don't want to waste their time. Take a look at the issue TOO EARLY AUTOMATION.
4. Testers don't understand that automation can also ease their work with manual tests. The issue INADEQUATE COMMUNICATION will show you what patterns can help you in this case.
5. Testers have been burned before with test automation and don't want to repeat the experience. Look up issue UNMOTIVATED TEAM for help here.
6. Testers do see the value of automation, but don't want to have anything to do with it. Your issue is probably NON-TECHNICAL TESTERS.
7. Supporting automation is not in the test plan and so testers won't do it. Check the issue AD-HOC AUTOMATION for suggestions.

Liz considered the various possible answers. “Ok, I don’t think that number 1 is the case, but numbers 2 or 5 could be. Four doesn’t sound probable and I don’t believe in number 7 either. Wait a minute: I think number 6 is the right one! I have been told that the testers themselves have asked for automation, but when I asked them something about automation I got the answer that they are not developers. The issue *NON-TECHNICAL TESTERS* sounds right to me”.

“By the way” she asked, “Why are the issues written in italic capitals?”

“It’s a convention in the wiki: the patterns are written in capital letters and the issues in italic capitals to tell them apart. In this way you can use their names almost as words in a kind of meta-language, a pattern language!”

“I see,” said Liz “Please go on to the *NON-TECHNICAL-TESTERS*.”

Jack nodded and clicked over to the issue:

***NON-TECHNICAL TESTERS* (Process Issue)**

Examples

1. Testers are interested in testing and not all testers want to learn the scripting languages of different automation tools. On the other hand, automators aren’t necessarily well acquainted with the application, so there are often communication problems.
2. Testers can prepare test cases from the requirements and can therefore start even before the application has been developed. Automators must usually wait for at least a rudimentary GUI or API.

Resolving Patterns

Most recommended:

- DOMAIN-DRIVEN TESTING: Apply this pattern to get rid of this issue for sure. It helps you find the best architecture when the testers cannot also be automators.
- OBJECT MAP: This pattern is useful even if you don’t implement DOMAIN-DRIVEN TESTING because it forces the development of more readable scripts.

Other useful patterns:

- KEYWORD-DRIVEN TESTING: This pattern is widely used already, so it will be not only easy to apply for your testers, but you will also find it easier to find automators able to implement it.
- SHARE INFORMATION: If you have issues like Example 1. this is the pattern for you!
- TEST AUTOMATION FRAMEWORK: If you plan to implement DOMAIN-DRIVEN TESTING you will need this pattern too. Even if you don’t, this pattern can make it easier for testers to use and help implement the automation.

"Oh" said Liz "I thought that we would get an answer and not just a link to other patterns!"

Jack nodded. "The test automation patterns follow their own rule to put information in only one place. I think that's the reason they were written in a wiki in the first place!"

"Another question:" said Liz, "what is the difference between the most recommended and the other useful patterns?"

"At the tutorial they explained that you should always look first at the most recommended patterns, but if for some reason you cannot apply them, then you should at least apply one or more of the useful ones. Here the most recommended pattern is DOMAIN-DRIVEN TESTING. Do we start with that one?"

"Yes, of course" replied Liz, even as she was eying the pattern TEST AUTOMATION FRAMEWORK.

DOMAIN-DRIVEN TESTING (Design Pattern)

Description

Testers develop a simple domain-specific language to write their automated test cases with. Practically this means that actions particular to the domain are described by appropriate commands, each with a number of required parameters. As an example, let's imagine that we want to insert a new customer into our system. The domain-command will look something like this:

New_Customer (FirstName, LastName, HouseNo, Street, ZipCode, City, State)

Now testers only have to call New_Customer and provide the relevant data for a customer to be inserted. Once the language has been specified, testers can start writing test cases even before the SUT has actually been implemented.

Implementation

To implement a domain-specific language, scripts or libraries must be written for all the desired domain-commands. This is usually done with a TEST AUTOMATION FRAMEWORK that supports ABSTRACTION LEVELS.

There are both advantages and disadvantages to this solution. The greatest advantage is that testers who are not very adept with the tools can write and maintain automated test cases. The downside is that you need developers or test automation engineers to implement the commands so that testers are completely dependent on their "good will". Another negative point is that the domain libraries may be implemented in the script language of the tool, so that to change the tool may mean to have to start again from scratch (*TOOL DEPENDENCY*). This can be mitigated to some extent using ABSTRACTION LEVELS.

KEYWORD-DRIVEN TESTING is a good choice for implementing a domain-specific language: Keyword = Domain-Command.

Potential problems

It does take time and effort to develop a good domain-driven automated testing infrastructure.

“Aha!” Liz spluttered, relieved. “After all, the pattern I need is really TEST AUTOMATION FRAMEWORK! That’s exactly what I wanted to do all the time! Let’s have a look at this pattern then.”

TEST AUTOMATION FRAMEWORK (Design Pattern)

Description

Using or building a test automation framework helps solve a number of technical problems in test automation. A framework is an implementation of at least part of a testware architecture.

Implementation

Test automation frameworks are included in many of the newer vendor tools. If your tools don’t provide a support framework, you may have to implement one yourself.

Actually, it is often better to design your own TESTWARE ARCHITECTURE, rather than adopt the tool’s way of organising things - this will tie you to that particular tool, and you may want your automated tests to be run one day using a different tool or on a different device or platform. If you design your own framework, you can keep the tool-specific things to a minimum, so when (not if) you need to change tools, or when the tool itself changes, you minimise the amount of work you need to do to get your tests up and running again.

The whole team, developers, testers, and automators, should come up with the requirements for the test automation framework, and choose by consensus. If you are comparing two frameworks (or tools) use SIDE-BY-SIDE to find the best fit for your situation.

A test automation framework should offer at least some of the following features:

- Support ABSTRACTION LEVELS.
- Support use of DEFAULT DATA.
- Support writing tests.
- Compile usage information.
- Manage running the tests, including when tests don’t complete normally.
- Report test results.

You will have to have MANAGEMENT SUPPORT to get the resources you will need, especially developer time if you have to implement the framework in-house.

“Hmm” mused Liz “This pattern tells me what such a framework should do, but actually I already have plenty of ideas myself. What I expected was some suggestions about how to do it!”

"You are right," answered Jack. "Maybe if you delve down into the detail of some of the other patterns, you will find more advice about that, particularly things like ABSTRACTION LEVELS and TESTWARE ARCHITECTURE".

"Well, hopefully, I will find things that I have already implemented and maybe some new ideas when I look at the detail. But just having a good technical framework isn't going to work here, if all the people seem to have no time for progressing the automation!"

"Good point! But notice that this pattern also suggests getting MANAGEMENT SUPPORT. Let's take a look at that."

"Well I already tried that by going to Tom and look where that got me!" countered Liz.

"Shall we just see what it suggests besides having one short meeting?"

"Yes, you're right again, Jack - let's have a look."

MANAGEMENT SUPPORT (Management Pattern)

Description

Many issues can only be solved with good management support.

When you are starting (or re-starting) test automation, you need to show managers that the investment in automation (not just in the tools) has a good potential to give real and lasting benefits to the organisation.

Liz commented here: "From the little I've seen of their current automation, it looks like I will be starting or re-starting with my type of automation, not trying to prop up what they already have, which is giving them so many problems."

"I agree with that," said Jack. "Let's see what it suggests."

Implementation

Some suggestions when starting (or re-starting) test automation:

- Make sure to SET CLEAR GOALS. Either review existing goals for automation or meet with managers to ensure that their expectations are realistic and adequately resourced and funded.
- Build a convincing TEST AUTOMATION BUSINESS CASE. Test automation can be quite expensive and requires, especially at the beginning, a lot of effort.
- A good way to convince management is to DO A PILOT. In this way they can actually "touch" the advantages of test automation and it will be much easier to win them over.
- Another advantage is that it is much easier to SELL THE BENEFITS of a limited pilot than of a full test automation project. After your pilot has been successful, you will have a much better starting position to obtain support for what you actually intend to implement.

Liz considered the suggestions. "I don't know what their goals are for automation - I wonder if they ever thought about that. But clearly, they did want automation, so

they must have had some idea what they wanted. I'll try to find out about that but challenging their automation goals probably isn't the best way to get the help I need!"

Jack laughed, "Yes, going in and telling them they've done it all wrong probably isn't the best way to build new relationships with the people."

"Building a business case isn't relevant here," Liz said. "Maybe DO A PILOT is a more realistic idea, but I thought that you only do a pilot when you are starting something, not in the middle of things!"

"Not necessarily," replied Jack "let's look it up before you start complaining."

"OK, let's do that."

DO A PILOT (Management Pattern)

Context

This pattern is useful when you start an automation project from scratch, but it can also be very useful when trying to find the reasons your automation effort is not as successful as you expected.

Description

You start a pilot project to explore how to best automate tests on your application. The advantage of such a pilot is that it is time boxed and limited in scope, so that you can concentrate in finding out what the problems are and how to solve them. In a pilot project nobody will expect that you automate a lot of tests, but that you find out what are the best tools for your application, the best design strategy and so on.

You can also deal with problems that occur and will affect everyone doing automation and solve them in a standard way before rolling out automation practices more widely. You will gain confidence in your approach to automation. Alternatively, you may discover that something doesn't work as well as you though, so you find a better way - this is good to do as early as possible! Tom Gilb says: "If you are going to have a disaster, have it on a small scale"!

Implementation

Here some suggestions and additional patterns to help:

- First of all, SET CLEAR GOALS: with the pilot project you should achieve one or more of the following goals:
 - Prove that automation works on your application.
 - Choose a test automation architecture.
 - Select one or more tools.
 - Define a set of standards.
 - Show that test automation delivers a good return on investment.
 - Show what test automation can deliver and what it cannot deliver.
 - Get experience with the application and the tools.

- Try out different tools in order to select the **RIGHT TOOLS** that fit best for your SUT, but if possible **PREFER FAMILIAR SOLUTIONS** because you will be able to benefit from available know-how from the very beginning.
- Do not be afraid to **MIX APPROACHES**.
- **AUTOMATION ROLES**: see that you get the people with the necessary skills right from the beginning.
- **TAKE SMALL STEPS**, for instance start by automating a **STEEL THREAD**: in this way you can get a good feeling about what kind of problems you will be facing, for instance check if you have **TESTABLE SOFTWARE**.
- Take time for debriefing when you are through and don't forget to **LEARN FROM MISTAKES**.
- In order to get fast feedback, adopt **SHORT ITERATIONS**.

What kind of areas are explored in a pilot? This is the ideal opportunity to try out different ways of doing things, to determine what works best for you. These three areas are very important:

- Building new automated tests. Try different ways to build tests, using different scripting techniques (**DATA-DRIVEN TESTING** or **KEYWORD-DRIVEN TESTING**). Experiment with different ways of organising the tests, i.e. different types of **TESTWARE ARCHITECTURE**. Find out how to most efficiently interface from your structure and architecture to the tool you are using. Take 10 or 20 stable tests and automate them in different ways, keeping track of the effort needed.
- Maintenance of automated tests. When the application changes, the automated tests will be affected. How easy will it be to cope with those changes? If your automation is not well structured, with a good **TESTWARE ARCHITECTURE**, then even minor changes in the application can result in a disproportionate amount of maintenance to the automated tests - this is what often "kills" an automation effort! It is important in the pilot to experiment with different ways to build the tests in order to minimise later maintenance. Putting into practice **GOOD PROGRAMMING PRACTICES** and a **GOOD DEVELOPMENT PROCESS** are key to success. In the pilot, use different versions of the application - build the tests for one version, and then run them on a different version, and measure how much effort it takes to update the tests. Plan your automation to cope the best with application changes that are most likely to occur.
- Failure analysis. When tests fail, they need to be analysed, and this requires human effort. In the pilot, experiment with how the failure information will be made available for the people who need to figure out what happened. What you want to have are **EASY TO DEBUG FAILURES**. A very important area to address here is how the automation will cope with common problems that may affect many tests. This would be a good time to put in place standard error-handling that every test can call on.

Potential problems

Don't bite off more than you can chew: if you have too many goals you will have problems achieving them all.

Do the pilot on something that is worth automating, but not on the critical path.

Make sure that the people involved in the pilot are available when needed - managers need to understand that this is "real work"!

"Wow - that's a really substantial pattern." said Liz. "And you were right: it can also be used 'in the middle of things'". And she pointed to the topic *Maintenance of automated tests*. "Awesome! Now, the problem is how to implement it." she reflected. "If I start with new test cases, which sounds sensible, I have no idea what to test and I'm not going to get any support. On the other hand, if I start by refactoring some of the old ones, where I can understand what a test does just by executing it, I'll be in a scrape because I will not be doing the job I was hired for! I guess I'm stuck between a rock and a hard place!"

Now it was Jack's turn to laugh. "Yes, I can see your point!" he added "but maybe you can do both!"

"What do you mean?" asked Liz, curious "How am I going to get away with that?"

"Look, they will soon give you some test cases to automate. For a start, select one that uses features that are already used in the other automated test cases. If you want to show in the pilot that one should use GOOD PROGRAMMING PRACTICES, then you will have to refactor some of them just to be able to reuse what has already been automated!"

"Actually," answered Liz, "I did look at some of the existing automated tests even though I was told to leave them to the other testers, and it looks like they were not very well designed at all. They told me they were keyword-driven, but their tests had lots of repetition and weren't modular - I could see that even by just looking at a few of them."

"That's the solution then!" exclaimed Jack. "You will be improving their existing tests and making them more modular in order to use them to do the new tests that you were assigned, like going in through the 'back door'."

"That's a good idea, I might be able to do that and explain it afterwards!" said Liz. "But then I won't be able to go on with my new framework yet."

Jack said, "As the saying goes, 'Forgiveness is easier than Permission'. But as to the framework you want to build, remember what it said as a potential problem: *Don't bite off more than you can chew: if you have too many goals you will have problems achieving them all*. I think that right now you must first convince them to listen to you."

Liz wasn't very happy with this conclusion. "Well, I don't really like it, but I realize that you're right! Again! I guess I am too focused on doing the framework that I want to do and know will be best long-term, but I need to take them with me one

step at a time. I think it's a great idea, to sneak in at their existing automation by the new tests that I will be automating - I'll let you know how it works. Why don't you come to me for lunch? When would you have time? What about next week?"

"Hmm, I have to ask the boss," Jack said chuckling and called Deirdre.

"Darling, how about going over to Liz next weekend?"

"Next week it's my brother's birthday. How about the week after that?" answered Deirdre.

"Oh, no," said Liz, "that week-end I'll be playing away with the band and won't be home. What about a week later? And if you have time why don't you come to hear us? I can get you the best seats!" she added with an inviting smile.

They both said "Sure! We'll be there! First, to the concert and the week after for lunch!"

In the meantime, Deirdre's specialty lentil soup³ was ready, so they all moved over to the table. Both Jack and Deirdre loved Liz's tiramisu, and Deirdre especially appreciated the alcohol-free portion for her. "Can we have your recipe?" asked Deirdre. "Only on one condition," said Liz, "if you give me your recipe for the wonderful lentil soup!" "OK," said Deirdre, "but be warned, it makes an enormous amount!"

Later, drinking coffee, Liz remarked with an exaggeratedly disapproving grimace "I really appreciate your help Jack, but you are annoyingly correct too much of the time".

Deirdre giggled "Oh I quite agree with you – it is annoying that he's always right!"

Liz was a bit envious as the two fondly looked into each other's eyes.

³ See attachment for recipe

1.1.3. Liz makes some changes

Re-Writing keywords?
GOOD PROGRAMMING PRACTICES
is the right pattern!

On Monday morning Liz had so many ideas about how to get test automation moving at Digiphonia that she was really looking forward to going to work.

She had decided to ask Ronald for some test cases to automate, but first she headed to the kitchen to get coffee. Just as she was filling her cup, Paul walked in and was delighted to find her there.

“Hello!” he said, “what a nice coincidence! I was just looking for you!”

Liz looked at him surprised and said, “Hi Paul, what can I do for you?”

“Is it this week-end that your jazz band is playing? The percussionist from the Beatles would like to come too, and that, with our wives, makes four of us. You had promised to get us good seats.” he added.

Liz beamed “Awesome, I’m sure you’ll love it, but it’s not this week-end, it’s the next one on Saturday. I’ll arrange it. By the way, after the concert we usually all go out to a restaurant; why don’t the four of you come along?”

“Why, that’s a splendid idea,” Paul said, and continued, “and how are you getting on with your test automation?”

Now Liz was kind of abashed and was not sure what she should tell the old gentleman. He noticed her restraint and taking her arm gallantly led her to his office. “You know, my dear,” he started, “I must give you some advice. Do you have a few minutes for me?”

Now Liz was totally baffled and just nodded.

“What I want to say,” continued Paul unruffled, “is that you should take your time with our test automation. All these smart people here work like crazy without ever stopping to reflect on what they are doing. I suspected that our automation effort would come to a sorry end ever since I saw how Ronald started automating everything without looking either left or right!”

Liz chuckled at the idea and Paul went on “You are also a musician and we know that to reach perfection you must take your time, must try out different possibilities and practice a piece again and again until it just flows. I don’t think that automating tests is much different, so please take your time, explore different strategies and don’t be afraid to try this or that.” Finally, giving her his friendliest smile and a wink, he added. “Please promise! From musician to musician!” and when she smiled back and agreed, he let her go.

“Awesome,” thought Liz, “Paul has actually suggested that I DO A PILOT! Now I call that motivation!” and she went back to her desk to have another look at what the pattern said about maintenance of automated tests.

DO A PILOT (Management Pattern)

Maintenance of automated tests.

When the application changes, the automated tests will be affected. How easy will it be to cope with those changes? If your automation is not well structured, with a good TESTWARE ARCHITECTURE, then even minor changes in the application can result in a disproportionate amount of maintenance to the automated tests - this is what often "kills" an automation effort! It is important in the pilot to experiment with different ways to build the tests in order to minimise later maintenance. Putting into practice GOOD PROGRAMMING PRACTICES and a GOOD DEVELOPMENT PROCESS are key to success. In the pilot, use different versions of the application - build the tests for one version, and then run them on a different version, and measure how much effort it takes to update the tests. Plan your automation to cope the best with application changes that are most likely to occur.

"Hmm" she thought "I can't just change the testware architecture all by myself, but, as Jack proposed, I can use GOOD PROGRAMMING PRACTICES! I wonder if the suggestions are much different from developing normal software," and she immediately clicked on the appropriate link.

GOOD PROGRAMMING PRACTICES (Process Pattern)

Use the same good programming practices for test code as in software development for production code.

Context

This pattern is appropriate when you want your automation scripts to be reusable and maintainable, that is when your test automation is to be long lived.

Description

Scripting is a kind of programming, so you should use the same good practices as in software development.

Implementation

If you don't have an automation engineer, have developers coach the automation testers. Important good practices are:

- DESIGN FOR REUSE: Design reusable testware.
- KEEP IT SIMPLE: Use the simplest solution you can imagine.
- SET STANDARDS: Set and follow standards for the automation artefacts.
- SKIP VOID INPUTS: Arrange for an easy way to automatically skip void inputs.
- INDEPENDENT TEST CASES: Make each automated test case self-contained.
- ABSTRACTION LEVELS: Build testware that has one or more abstraction layers.

- Separate the scripts from the data: use at least DATA-DRIVEN TESTING or, better, KEYWORD-DRIVEN TESTING.
- Apply the DRY Principle (Don't Repeat Yourself), also known as DIE (Duplication is Evil).

Put your best practices in a Wiki so that both testers and developers profit from them.

Potential problems

If the software developers don't use good programming practices, then don't follow them, but investigate good programming outside of your company. (Read books!).

"OK, let's see where it would be best to start" reflected Liz. "DESIGN FOR REUSE is obvious and so is KEEP IT SIMPLE. SET STANDARDS is something for later, just as ABSTRACTION LEVELS. Ah, I got it. They have been talking about KEYWORD-DRIVEN TESTING: that's where I should start. I'll design some nice new keywords!"

And she got up, but then had an afterthought, "I think I should take a look at what the wiki has to offer about keywords, after all their suggestions have been quite helpful so far," and she went back to the wiki and typed "Keyword" into the search box to find the pattern.

KEYWORD-DRIVEN TESTING (Design Pattern)

Tests are driven by keywords that represent actions of a test and may include input data and expected results.

Context

This pattern is appropriate:

- When you want to write test cases that are practically independent from the Software under Test (SUT). If the SUT changes, the functionality behind the keyword must be adapted, but most of the time the test cases themselves are still valid.
- When testers should be able to write and run automated tests even if they are not adept with the automation tools.
- If you want testers to start writing test cases for automation before the SUT is available to test.

Description

Keywords are the verbs of the language that the tester uses to specify tests, typically from a business or domain perspective. A keyword specifies a sequence of actions together with any required input data and/or expected results.

Keywords are most powerfully used at a high level, representing a business domain. Different domains would have different keywords. High level keywords for an insurance application, for example, might include "Create New Policy", "Process Claim" or "Renew Policy". High level keywords for a mobile phone, for example, might include "Make Call", "Update Contact" or

"Send Text Message". There may be some keywords that are common across more than one domain, particularly at lower levels, such as "Log In" and "Print Page".

Implementation

Each keyword is processed by an associated script, which may call other reusable scripts from a library. A keyword script can be written in a common coding language, as calls to other keywords or in the scripting language of the tool. It is a good idea to keep the tool-specific scripts to a minimum as this will help to reduce script maintenance costs. The keyword script reads and processes the input data for the keyword, and/or checks the expected output. The implementation of the architecture of the keywords is often referred to as a TEST AUTOMATION FRAMEWORK, which determines the choice of scripting language and how composite keywords are defined.

Potential problems

Take care that the keywords describe a clear and cohesive action. If the keywords build the "words" for a domain specific language, as in DOMAIN-DRIVEN TESTING, your testers will easily be able to write automation test cases and will be able to start even before the SUT is available for running tests.

Another problem can come up when you need too many parameters (input data) for a given keyword: it becomes unwieldy if you have to scroll repeatedly in order to see or input all the parameters. Break the unwieldy keyword into shorter independent functions (which can call and be called from the others).

Liz read the pattern attentively and was reassured to find that the suggestions in the pattern were things that she already understood, so she was now more confident that she had the right idea to go forward. Having now decided where to start, she went to see Ronald and got an excel file with test cases to be automated. She also asked for, and got, the excel files of the test cases that had been already automated. "Because" she explained "If I find similar actions then I know where to find the appropriate code and I can use what you have already automated as a template".

She began to compare the new test cases with the existing automation as Jack had suggested. It took some more days of research until finally she found just the right thing: all the Digiphonia Screech apps had the same core and one of its most important functionalities was setting up the parameters for a new composition. However, this was not only one of the features that changed most often, but also one that was used in all test cases. Liz believed that if she could refactor it to a keyword that could be called from all the test cases, the maintenance work would be reduced substantially.

"Awesome" she said to herself "If testers and management then still don't believe in GOOD PROGRAMMING PRACTICES, and this is really only a tiny example, then I'd better look for another job!"

During her next lunch break she considered how to do it most efficiently “Would it be better to code the script from scratch or to just adapt it from existing code?” She had noticed that the code was not always identical, even if it supposedly did the same thing so she had to look at all the possible variations in order to be sure to really develop a keyword for all uses. Another thing she had noticed was that having implemented the test cases using the capture facility of the tool, the testers had given no thought to how AutomatePro accessed the GUI-elements they were working with. As a default, the tool had often chosen labels, indices or even positions as criteria for recognizing an element, and Liz was not surprised to see that this had been one of the main reasons for the increasing maintenance costs. When the developers changed something in the GUI, the tool more often than not could not recognize the elements any longer and the script would have to be recorded anew.

“This is strange” she thought “The wiki didn’t mention this problem in the pattern GOOD PROGRAMMING PRACTICES. Did they forget it? Then it’s high time to add it!”

She went back to the page for GOOD PROGRAMMING PRACTICES: no mention! “Hmm” she reflected “Maybe it’s hidden in one of the suggested patterns...” She opened DESIGN FOR REUSE.

DESIGN FOR REUSE (Design Pattern)

Description

Design modular scripts: write script code for some functionality in only one place and call it when needed. In this way you can reuse the components again and again and if you do need to change something you have to implement and test it only in one place.

Reuse test data as much as possible.

Implementation

- When you see that you need to use again a part of some script (keyword), extract it and build with it a new script (keyword) that you can then call from the original script and from any future scripts (keywords) that may need it. Be sure to document exactly what it does and what parameters must be passed and give it a name that makes it easy to find.
- **SINGLE PAGE SCRIPTS:** write a script for every screen you want to automate, where you include all the GUI-objects on the screen. The input parameters can be used to communicate which GUI-objects will be activated in the specific test case and how they will be activated. For example, if a parameter is missing, the GUI-object will be ignored.
- Implement an **OBJECT MAP**: if the objects get meaningful names it will be easier for the testers to understand the scripts. Also, in this way you pave the way for your scripts to achieve **TOOL INDEPENDENCE**.
- Write a script for all the various set-ups that you will need. Again, define with input parameters the specifics for each test case.

- Generate test data that can be reused repeatedly (for instance a customer record with some specific characteristics).
- DOCUMENT THE TESTWARE: give the “building blocks” (scripts or data) descriptive names so that a tester can see at a glance what is available, how to use it and when to use it.

Recommendations

If possible, get a developer to coach you in GOOD PROGRAMMING PRACTICES.

Liz quickly read through the pattern. “Aha” she said to herself “There it is, OBJECT MAP!” and immediately followed the link.

OBJECT MAP (Execution Pattern)

Pattern summary

Declare all the GUI-Objects in the Object Map of the test automation tool.
(Or define your own if appropriate.)

Context

This pattern is appropriate if you want your test automation to be long lasting, because your scripts will be easier to read and less dependent on the current tool.

This pattern isn't needed if you only write disposable scripts or if your application is exceptionally stable, with object names never or hardly ever being changed.

Description

Declare all the GUI-Objects in the Object Map of the tool (note: this may also be called a GUI map or Object Repository). You can then write your scripts using standardized names. The scripts will be more readable and if you have to migrate to another tool you will not need to change the scripts to allow for new names even if tools seem never to use compatible naming conventions.

Implementation

This is one way to implement TOOL INDEPENDENCE as your tests will be one layer removed from the specific names in the software.

Some suggestions:

- Ask the developers to use unique names to define the GUI-Objects. If the same objects are used over a number of different screens (object-oriented inheritance) it may be useful to add the screen name to the object name for better recognition.
- When driving the GUI avoid using pixels, screen coordinates or bitmaps to identify graphical objects. Screen coordinates can vary from machine to machine, from operating system to operating system or from browser to browser. Also changes in the Software under Test (SUT) can mean different positions for the GUI-Objects, so your scripts are more likely to break.

- If the SUT runs in different countries, you must also avoid using text labels if you don't want to write new scripts or have to have a separate Object Map for every new language!
- If you can't use the tool's facilities, you can make your own "translation table", in which you have the names you will use in the tests in one column, and the current name used by the software in another column. Then if the software names change, you only have to update your translation table and all your tests will continue to work.

Some suggestions for object naming:

- Internal developer name: if the developers follow a style guide it can be useful to use the same names. In this way you also have the advantage of being able to communicate with the developers in their own "language".
- Label on screen: using the labels on the screen makes it easier for testers to follow what happens in the script.
- Functional term: using the correct terms is a useful way to document what the script is supposed to achieve.

Potential problems

This approach might be perceived as an unnecessary additional complication, but it is needed to gain a level of abstraction from the tool.

"Awesome," thought Liz, "It's all there! I wonder how much work must have gone in implementing this wiki!"

"But I will have to rework all the GUI-elements," she thought. "For now, I'll do only the ones that I'll be using right away, so now I just need to do the parameter input."

After having checked how AutomatePro handled OBJECT MAP she updated the GUI elements that she needed. Now she had to implement the keyword, and that turned out to be more difficult and time-consuming than she had expected. She wanted to develop one keyword, but she realized that the differences between the different test cases were mainly due to the different parameters they were using. Trying to put everything in one keyword produced an unreasonably long parameter list.

"Hmm" she thought "The second pattern recommended in GOOD PROGRAMMING PRACTICES was one called KEEP IT SIMPLE, but it's not that easy! I'd better break this keyword into smaller ones where each collects just a few parameters and checks them. Then each will call the same core keyword to set them."

Full of enthusiasm, Liz developed the new keywords. To test them, she built them into a few specially chosen old test cases. It took hours, (actually days) but she really enjoyed it. She thought all the time about the reaction from Jill, Tim and Ronald when they realized that the new release wouldn't create the usual chaos.

Friday as she cleared up her desk, she knew that she was not finished yet, but she felt that that she had made a big step forward and that for this week she had really earned her wage.

1.1.4. Abstraction Hamburgers and Keyword Furniture

Eat a hamburger,
assemble some furniture:
explaining patterns!

On Monday Liz, having finished implementing and testing the new keywords, began inserting them in all the ‘old’ test cases. On Wednesday she was finally through and the new version of the app was also ready for the final automated test before the release.

Before going home, Ronald checked that the automated tests were started. He was not looking forward to telling his wife that the next day he would probably have to stay late again. She had been quite understanding so far, but Ronald knew that she hated it when he worked so late. But he expected to be busy fixing hundreds of automated tests that would have failed, as usually happened, and there was great pressure to get it done on Thursday.

While they cleared up their desks Jill and Tim shared jokes about the new “bugs” they would have to remove from the scripts the next day, probably continuing over the weekend as usual. Liz hadn’t realised that they usually had to work through the weekend to clear up problems in the automated tests.

The next day the automated tests were done. Liz tried not to show it, but she was really excited. Would her “pilot” be able to convince testers and management?

Ronald checked the results and as usual most of the red dots were “automation bugs”. He sighed and opened the first script. Now he noticed the changes Liz had made and at first was quite angry with her. “Don’t we have enough work without you messing around? You are supposed to automate new test cases. Leave these alone!” he cried. The next test case was similarly “messed up” and the next and the next. But then he realised that all of them had actually had the same problem, in one of the new keywords. Then he got it: he only needed to update the one keyword script that all the test cases were calling. He still didn’t really believe that it could be so easy, but he shut up, updated the one script and ran a couple of tests again. It worked! He went over to Liz.

“Liz,” he said “I’m sorry, how did you do it? When did you do it? I saw that you were working on the new test cases, and I know you mentioned something about changing the existing tests, but I had no idea it could make such a difference.”

Liz just smiled and said, “GOOD PROGRAMMING PRACTICES!”

At that moment Tim came through the door and exclaimed “Isn’t that one of the patterns we should do?”

“Should have done” corrected Ronald “and now Liz is implementing it for us!”

Liz was surprised to hear that Tim recognized the pattern “You’re right,” she said, “but if you know the test automation patterns why you didn’t use them?”

Tim looked pained. "I learned about them a couple of months ago at the Galaxy Test Conference. But by then we had already automated all this stuff!"

"After Tim told us about them, we used the Diagnostic feature in the wiki to find a solution," added Jill, "but the suggestions were too technical for us and, as you now know, that's when we realized that we needed a 'real' automator."

"Too technical?" Liz asked, puzzled "What for instance?" and she thought: "this is the chance I was waiting for: helping others has always been the best way to get help myself!"

The testers started talking all together. Putting up her hands Liz asked "Please one at the time! Tell me about the diagnostic."

Tim started "Well, I had got some instructions at the conference, but we didn't really know how to use it. Each one of us wanted to select a different answer."

Then Jill butted in, "Finally we agreed on the issue *BRITTLE SCRIPTS* and looked up what patterns were suggested."

"Wait a moment," interrupted Liz, "let's look it up now, then you can show me where you found it confusing."

***BRITTLE SCRIPTS* (Design Issue)**

Issue Summary

Automation scripts have to be reworked for any small change to the Software Under Test (SUT).

Examples

Scripts are created using the capture functionality of an automation tool. If in the meantime something has been changed in the application, the tests will break unless recorded anew.

A small change to the application (such as moving something to a different screen or changing the text of a button) causes many scripts to fail because this information is embedded in the scripts for many tests.

Questions

How do you develop automation scripts? (Capture is not good long-term though ok for short complex actions).

Is there repeated code in any scripts? (Keep your automation code DRY - Don't Repeat Yourself - put common code and actions into scripts called by other scripts).

What kinds of changes are most likely to happen to the application? (Make your automation most flexible for those changes).

Resolving Patterns

Most recommended:

- **TESTWARE ARCHITECTURE:** This pattern encompasses the overall approach to the automation artefacts and is best thought about right

from the start of automation so that you can avoid having *BRITTLE SCRIPTS*. This pattern is implemented using:

- **ABSTRACTION LEVELS**: This is the pattern to apply if you want to delegate some of the maintenance effort to the testers. It will enable you to write test cases that are both independent from the SUT and from the technical implementation of the automation (including the tool(s)).
- **MAINTAINABLE TESTWARE**: This is the pattern to apply if you want to get rid of the issue once and for all. If you haven't implemented it yet, you may want to apply at least some aspects of this pattern.
- **MANAGEMENT SUPPORT**: This is the pattern to apply if you are missing support or resources that you need to develop **MAINTAINABLE TESTWARE**.
- **MODEL-BASED TESTING**: This pattern involves considerable effort at the beginning but is the most efficient in the long run. Using a test model, the test cases can be cleanly separated from the technical details. Frequently used sequences of test steps can be defined as reusable and parameterized building blocks. If the SUT changes, usually only a few building blocks need to be adapted while the test scripts are updated automatically.
- **COMPARISON DESIGN**: Design the comparison of test results to be as efficient as possible, balancing Dynamic and Post-Execution Comparison, and using a mixture of Sensitive and Robust/Specific comparisons.

Other useful patterns:

GOOD PROGRAMMING PRACTICES: This pattern should already be in use! If not, you should apply it for all new automation efforts. Apply it also every time you have to change current testware.

Again, it was Jill who went on, “There,” she pointed on the screen. “We looked up **TESTWARE ARCHITECTURE** and landed on **ABSTRACTION LEVELS** and **DOMAIN-DRIVEN TESTING**.”

Tim continued “We weren’t comfortable with any of those, so we looked up **MAINTAINABLE TESTWARE**. That seemed to be more our thing.”

Jill added, “And there we were sent again to **ABSTRACTION LEVELS** and **DOMAIN-DRIVEN TESTING!** At that point we decided that we had taken the wrong path in the Diagnostic and decided that we actually needed help from an experienced automator,” and with a smile she finished, “and here you are!”

“Well,” said Liz, “I’ve been reading the patterns as well, and I can explain them to you. I can also tell you what patterns I have been using since I started here. I can even tell you what things have given me the most trouble!”

Ronald had just been listening, but at that point he stepped in “Hey, people, come back to Earth! We still have a lot of work to do! I am also interested, but we have to postpone it.”

Liz had a suggestion. "I understand," she said, "but why don't we go to lunch together and I could explain one pattern and how we could use it here. If we do that for a number of days, you'll gradually become more familiar with how they work."

"In the meantime," she added gingerly, "I will go on automating new test cases and I promise to mess up the old ones only when I really can't avoid it".

That same day at lunch, Liz, Jill, Tim and Ronald were at the cafeteria of another company in their building, Software Forge. The main staples there were pizza, hamburger, French fries and occasionally a soup, so even Jill, a vegetarian, liked to eat there.

This time instead of the usual chatter they were listening to Liz.

Liz began, "So where do we start? You mentioned ABSTRACTION LEVELS. Would that be OK?"

The three nodded, so she asked Jill: "If you imagine that your tomato soup was an automated script, would it be easy to maintain?"

Jill just stared back, "Tomato soup?"

"Yes," Liz continued, "imagine that the app, the recipe, has changed and you have to put less salt in it. How easy is that?"

"Not so easy," murmured Jill with a grimace. "I could add water, but the soup wouldn't get better!"

"Exactly," answered Liz, "and that's what your 'old' test scripts look like!"

Tim and Ronald just rolled their eyes.

"Now look," she added, "you see my hamburger? This is a good example of ABSTRACTION LEVELS!"

Now Jill, Tim and Ronald looked up, really puzzled. "A hamburger? You kidding?" said Ronald.

"Let me explain," replied Liz. "Imagine again that the app, the recipe, has changed. Now you want a cheeseburger instead of a normal hamburger. What do you have to do?"

Tim grinned, "I think I got it, you just open the bun, add the cheese and that's it!"

"Yes," said Liz, "and it also works the other way around. Imagine that you do have a cheeseburger and you want a normal hamburger: you just have to remove the cheese!"

"The trick," she continued, "is that the hamburger consists of different parts that you just have to put together, just as I inserted generalized keywords in the scripts I 'messed up'. So, for instance, when I say cheese it could be very different

cheeses, but my hamburger would still be a hamburger. Or," she went on, "instead of beef you could have chicken or fish or whatever."

"And that's not all," she added. "You need a cook, an automator, to make a tomato soup script, but actually anybody can put together a hamburger - a keyword-driven script!"

"Tim?" she asked, "do you understand why?"

Tim hesitated, but then answered "I think it's because in preparing a hamburger, different people do different things; one is specialized in broiling the patty, another washes and cuts the salad, a third cuts the buns and finally somebody puts it all together. Everybody does what he or she can do best."

Liz threw him a big toothy smile: "Awesome!" Then she asked, "What are the abstraction levels in our hamburger?"

Without waiting for an answer, she continued, "Deep inside you have the patty, the technical level consisting of scripts that actually execute the test case. Scripts that must be written by somebody savvy in the scripting language of the tool."

"Salad and tomatoes and buns etc. make for the next level - that would be data, databases and so on. Here you may need developers or database experts to get your salad just right!" she added with a smile.

"Finally, you have the outer level, the finished hamburger - the test case - that must be defined by somebody who knows the testing specifications." And she looked earnestly at the three testers.

After a little pause to take a sip of her drink she added, "The charm of this method is that even if the customers -the app- are not there –implemented- yet, one can already start preparing the hamburgers -the test cases- that will be needed."

"Do you still think that this pattern is too technical?" she asked finally, while taking a big bite of her hamburger.

"Cool," beamed Jill, "I like your way of explaining test automation patterns!"

And Tim added "I will never be able to eat another hamburger without thinking about ABSTRACTION LEVELS!"

That evening in the commuter train, Jill was not reading her usual fantasy e-book, but had searched for ABSTRACTION LEVELS in the wiki. She thought, "I suspect that Liz was just making fun of us. Let's see what the pattern really says!"

ABSTRACTION LEVELS (Design Pattern)

Description

The most effective way to build automated testware is to construct it so that it has one or more abstraction layers (or levels). For example, in software development, the code to drive the GUI is usually separated from the code that actually implements the business functionality and also separated from the code that implements access to the database. Each part communicates

with the others only through an interface. In this way each can be individually changed without breaking the whole as long as the interface is not changed (this is the theory, it is not always practiced).

For test automation this means that the testware is built so that you write in the scripting language of the tool only technical implementations (for instance scripts that drive a window or some GUI-component of the SUT). This is the lowest layer. The test cases call these scripts and add the necessary data. This is the next layer. And if you implement a kind of meta-language you get another layer. As for software code, the charm of abstraction layers is that since the layers are independent of each other they can be substituted without having to touch the other layers. The only thing that has to be maintained is the interface between them (how the test cases are supposed to call the scripts).

By separating the technical implementations in the tool's scripting language from the functional tests, you can later change automation tools with relative ease, because you will only need to rewrite the tool-specific scripts. Also if you keep the development technicalities apart from the test cases, even testers with no development knowledge will be able to write and maintain the testware. And they can start writing the tests even before the SUT has been completely developed. Another advantage is that you can reuse the technical scripts for other test automation efforts.

Implementation

There are different ways to implement abstraction layers. Which to choose depends on how evolved your test automation framework is.

- DATA-DRIVEN TESTING means that you separate the data from the execution scripts (drivers). In this case you must take care to correctly pair the data to the drivers.
- In KEYWORD-DRIVEN TESTING you specify a keyword that controls how the data is to be processed. Generally keywords correspond to words in a domain-specific language (for instance insurance or manufacturing). Normally used for DOMAIN-DRIVEN TESTING.
- In MODEL-BASED TESTING you create a test model of the SUT. Typically, the modelling of test sequences starts at a very abstract level and is later refined step by step. From the model, a generation tool can automatically create test cases, test data, and even executable test scripts.
- Use TOOL INDEPENDENCE to separate the technical implementation that is specific for the tool from the functional implementation.

Potential problems

Building a good TESTWARE ARCHITECTURE takes time and effort, and should be planned from the beginning of an automation effort. There is a temptation (sometimes encouraged by management) to "just do it". This is fine as a way of experimenting and getting started, for example if you DO A PILOT. However, you soon find that you have many tests that are not well structured and get STALLED AUTOMATION but now you are "locked in" to the wrong solution if you are not aware of the importance of abstraction levels.

Jill was convinced. Of course, Liz had simplified things, but now she did have a much better comprehension of the pattern than when she, with the others, had read the description before.

"Still," she wondered, "when we automated the 'old' test cases, we were creating keywords, at least that's what they are called in AutomatePro, and now I read that KEYWORD-DRIVEN TESTING is recommended...How can that be wrong?"

She decided, "Tomorrow I have to ask Liz about it. Something just doesn't fit."

The next day Liz was working on one of her new test cases when Jill came over with her cup of coffee.

"I've been thinking" she said "about KEYWORD-DRIVEN TESTING and I must admit that I'm confused. Could we talk about it at lunch?"

"Sure" replied Liz "let's all go together again if we can."

This time the cafeteria belonged to WWG (World Wide Goods) and there you got mostly Chinese or Indian food, but they were especially renowned for their dessert assortment.

At the table it was Jill that started talking:

"You know, Liz, I've been reading that pattern we spoke about yesterday, ABSTRACTION LEVELS, and I think I get it. What I don't understand though is that it suggests implementing KEYWORD-DRIVEN TESTING and that is actually what we did. Then why are we having all these problems? I just don't get it."

"Ah," answered Liz, "calling something a keyword doesn't mean that it is! Did you read the pattern?"

Jill looked down, "Well actually I didn't..."

"It's Ok," countered Liz "I can explain it."

"Ah," interrupted Tim looking at his curry, "but today we don't have hamburgers!"

Everybody laughed. "And we don't need any," continued Liz. "In order to understand keywords, we need another kind of example." She reflected a short while and then added "Some time ago I completely refurbished my home office with furniture from IKEA, a Swedish company".

"Well," she went on "it is really awesome how efficient their concept is: you just pick up a number of boxes and then assemble your stuff yourself at home. And lots of the parts are the same for a number of different objects."

Liz concluded "Keywords work just in the same way, but instead of assembling a table or a wardrobe you assemble a test case! The keywords are the building blocks. The problem with AutomatePro is that it calls the test cases themselves 'keywords'."

Tim interrupted her and completed “... and so each test case becomes a building block itself instead of being an assembly of building blocks!”

“Exactly,” approved Liz, “and that’s the reason you’ve had so much maintenance work!” She went on, “When the test cases are not built using real keywords, you repeat the same code again and again and if your app changes you must find all the repetitions and rework every single one of them!”

“Now I get it,” said Jill. “If you use the keywords, you have the code only once, so you only have to change it once!” and she looked at Ronald. “Now, I see why you were so enthusiastic the other day!”

Liz added “Another advantage of KEYWORD-DRIVEN TESTING is that you can use it even if you don’t know how to use the tool.”

“Oh, but we did get training,” interjected Tim.

“Yes,” replied Liz, “and so now you have to maintain the automation even when you don’t have time!” and then added, “Imagine that testers could just write or maintain the automated test cases by themselves. Wouldn’t that help share the work better?”

“Sure” said Jill “but how can we do that?”

“Ever heard of excel tables?” Liz asked innocently and added, “There are different ways to do it but as an example, you could write a keyword in one column, such as *Login* and in the next columns you would put the necessary parameters for the keyword. For *Login* they would probably be *User* and *Password*. On the next line you put the next keyword and so on until your test case is completed.”

“And,” she went on with enthusiasm, “if you choose readable names for the keywords, you can execute the test cases either automatically or manually, so you can run them even if the keywords haven’t been implemented yet.”

“Now, slow down,” interrupted Ronald. “The test cases have to be in the tool in order to be executed.”

“Yes, that’s what the people at AutomatePro want you to believe!” countered Liz. “But it doesn’t have to be so.” She added, “You need a simple framework to do it, but I could build you one easily. In fact, in my previous company I’ve done just that.” Then she thought, “maybe now I have them where I want them!”

“Hmm,” said Tim, “but just writing names and parameters on an excel sheet brings you only so far. Where do you get the actual keywords?”

“That’s where the ABSTRACTION LEVELS come in,” explained Liz. “Remember that one of the advantages was that each level is implemented by the respective experts. Testers can write the test cases and automators implement the keywords for the testers, so everybody does what he or she can do best.”

“I see,” said Jill, “but how do you know which keywords to build?”

Liz replied "I don't. Automation is team-work: you, the testers must tell me what you need, I must tell you what is feasible and so we can complement each other."

On the way back to the office everybody was silent, Liz was hoping that the seed she had planted for the framework would grow to a hardy plant and each one of the others was considering the advantages of such an approach for his or her work.

That evening Ronald, after reading the pattern KEYWORD-DRIVEN TESTING, searched the internet for keyword-driven testing and was amazed at the number of different ways it could be implemented. "We will have to think about it some more," he thought later as he sat on his sofa, his arm around his wife's shoulders, softly stroking the purring cats on his lap.

As for Liz, as soon as she had left the building she had shut off all thoughts about test automation and had started instead to look forward to the concert the next evening. She knew that Jack and Deirdre would enjoy it, but she was quite curious to see how Paul and his D-eatle pal would react.

"Well" she thought "I'll find out soon enough!"

1.1.5. Automation Trains and Object Map Desserts

What's for dessert?
Automated tests don't work
like manual ones!

On Monday Liz needed more time than usual to get off the ground. She was still reliving the concert and especially the socializing at the restaurant afterwards. Her jazz band and the two D-eatles pals had immediately chatted away as if they had known each other for ages, each appreciating the musical talents of the others. Even the non-musicians had taken a fancy to each other and had giggled together the whole evening!

After getting a cup of coffee she ordered herself, "OK, now get to work!" and started to examine once more the 'old' automated test cases. The week before, she had been busy refactoring them in order to insert the new keywords, but she had had the impression that something still didn't fit, and she wanted now to concentrate on finding what had caught her attention.

Just before lunchtime she finally got it. A lot of the test cases were building on each other so that they could only be run in the given sequence and would fail if one in the series before it would fail!

"Gotcha!" she said quite loud to herself and turning to the others said, "Are you coming for lunch? I have a new important lesson in test automation for you!"

They looked at her quizzically, but they all nodded.

Liz suggested to go again to the WWG cafeteria, as she needed to buy some toys, so at the entrance she said "Go on, I first have to get something here. I'll join you at the table."

The other testers went on and Liz entered the outlet shop and bought two tiny trains made from old soda cans - one red (Coke?) and one blue (Pepsi?). They were presents for Jack's boys but they would also do for her planned 'test automation lesson'!

Then she filled her tray and went to sit with the others. Seeing her trains, they became even more curious what she had in mind this time.

At last Jill couldn't resist any longer and said, "This time you'll use the trains instead of hamburgers, right?"

Liz laughed, "Yep," and added, "and since you started, tell me what you think these trains stand for."

Jill reflected, "Test cases?"

"Cold" said Liz.

Ronald burst in "Test suites?"

"Warmer" said Liz.

Tim added “Automated test suites?”

“Hot,” said Liz. “In that case what do the wagons represent?”

Now Jill repeated “The test cases?”

“Hot! Very hot!” replied Liz with a grin. “And do you notice something about these trains?”

All three looked baffled and so Liz went on, “These test cases are all depending on each other! If the second wagon in the red train burned up and therefore the following wagons got loose, you would have found an error - the burned-up wagon - but you wouldn’t have tested all the following test cases! This way of designing the test cases is fine for manual testing but is an absolute no-no for test automation! There is even an issue to describe it: *INTERDEPENDENT TEST CASES!*”

And looking earnestly at the three testers she added, “And you, as manual testers, have automated quite a lot of test cases like this.”

“Hmm,” said Jill after a short reflexion, “I believe you, but how do you avoid doing that? The test cases after all do build on each other!”

Liz smiled, “it all depends on the ‘engine’ - that is on how you set up the initial conditions. In the train you have one engine - the set-up - and the wagons - the test cases - are all in a specific sequence and run in that sequence.”

Now Tim interrupted her. “Ah” he exclaimed, “I think I got it: you need an engine for each wagon!”

“Awesome!” answered Liz trying to look serious “You just invented the car!”

“Actually,” countered Ronald as they all laughed. “A wagon with an engine would be more like a bus!”

Jill stepped into the discussion “Oh but think how much easier it would be to check a car-test case than a bus-test case! All the people! Not to speak of the luggage!”

They were all laughing heartily when Liz continued “Wait, I’ll show you the issue and then the appropriate patterns.”

She opened her back-pack, took out her tablet and showed them the following page:

INTERDEPENDENT TEST CASES (Design Issue)

Test cases depend on each other, that is, they can only be executed in a fixed sequence

Examples

Test cases must be executed in a fixed sequence because the preceding test cases create the initial conditions for the following ones.

Questions

Who designed the test cases, and for what purpose? Are manual test cases being automated “as is”?

If so, this can cause problems, because what makes sense for manual testing may not be the most appropriate solution for automated tests, particularly the dependencies of tests on each other. If one of a sequence of manual tests fails, the tester can usually find a way to continue testing, possibly by inserting the data that should have been produced by the failing test. But the tool will simply be “stopped in its track” by a failing test - it’s not possible or practical to anticipate all possible ways that all tests can fail so that an automated solution can be implemented.

See also [MANUAL MIMICRY](#) - an issue describing what happens when you try to automate manual tests too literally.

Resolving Patterns

Most recommended:

- [FRESH SETUP](#)
- [INDEPENDENT TEST CASES](#)

After they were done reading Liz said, already clicking on the appropriate link, “I’ll first show you the pattern INDEPENDENT TEST CASES and then the other one.”

INDEPENDENT TEST CASES (Design Pattern)

Make each automated test case self-contained

Context

This pattern is necessary if you want to implement long lasting and efficient test automation.

An exception is when one test specifically checks the results from the prior test (e.g., using a separate test to ensure data was written into a database and not just retrieved from working memory).

It is not necessary for just writing disposable scripts.

Description

Automated test cases run independently of each other, so that they can be started separately and are not affected if tests running earlier have failed. The test may consist of a large number of different scripts or actions, some to set up the conditions needed for a test, others to execute the test steps.

Automated tests should be short and well-defined. For example, if you have one long test that takes 30 minutes to run, maybe it fails after 5 minutes the first time, after 10 the next and after 15 the 3rd time. So far you have used half an hour and have 3 bugs (which you have fixed). But you are only half-way through the test. If you separate that test into 10 tests of 3 minutes each, you can start all of them. If you can run them in parallel, execution will only take 3 minutes. Maybe 3 or 4 of them fail, but you now know that all of the others have passed, so after you fix these, the whole set of tests should be passing, and it takes a lot less time.

Implementation

Every test starts with a [FRESH SETUP](#) before performing any action. Each test has proprietary access to its resources.

If a test fails (stops before completion), it must reset the Software under Test (SUT) and or the tool so that the following tests can run normally (see the pattern FAIL GRACEFULLY).

A self-contained test does NOT mean that just one test case tests the whole application! On the contrary each test should have ONE CLEAR PURPOSE derived from one business rule.

If you PRIORITISE TESTS, you will also be able to run or rerun the tests independently from each other.

Potential problems

Manual tests are often performed sequentially in order to spare set-up time. They should be redesigned before automating to make sure that the automation is as efficient as possible.

If the initial set-up is very complicated or takes too much time, then this pattern should not be used, instead use CHAINED TESTS.

After all were done reading, she flipped to the next pattern (which was also referred to from this one):

FRESH SETUP (Design Pattern)

Before executing each test prepares its initial conditions from scratch. Tests don't clean up afterwards

Context

Use this pattern for long lasting and maintainable automation, but it can be useful also when writing disposable scripts.

Description

Each test prepares its initial conditions from scratch before executing, so that it makes sure to run under defined initial conditions. In this way each test can run independently from all other tests. Tests don't clean up afterwards, so that if you want to check the results you can immediately control the state of the Software under Test (SUT).

Implementation

Initial conditions can be very diverse. Here some suggestions:

Database configuration:

1. Copy the table structure of the database for the current release of the SUT: in this way you will always have the current database. Since this may take some time you should do it only once at the beginning of a test suite.
2. Insert in the database the standard configuration data that you will need for each of the following test cases. This should also be done only once at the beginning of the test suite.
3. For each test case: make sure the relevant variable database entries are empty. If not, remove or initialise content (after ensuring that you have not wiped out the traces of prior test errors).

4. For each test case: insert the variable data that your test case expects.

File configuration:

Copy input or comparison files to a predefined standard directory.

SUT:

To be sure that the SUT is in the required state, it should be started anew for each test case and driven to the foreseen starting point.

Virtual machines:

For complex environments it may pay to start each time from a known VM snapshot. This has the added benefit that in the case of an erratic test then the failed VM can be automatically stored away for further analysis. Storage and time limitations would probably make this impractical for non-regression scenarios where you expect a lot of failures.

Leave the SUT as it is after the test is run. In this way, after running the test, you can immediately check the state of the SUT, database contents etc. without having to restart the test and stopping it before it cleans up. Even if you do have to restart it, e.g. because it was followed by other tests, you don't have to change the scripts in any way to repeat the test and check the results.

Potential problems

If the setup is very slow, you should consider using instead a SHARED SETUP.

After reading them, Ronald said, somewhat abashed, “I see that we still have a lot to learn!”

Liz smiled “That’s OK, if you are willing to learn, you will improve!” and then she added, “I feel like dessert, I’ll go get some. Just tell me what you would like.”

“Super!” said Jill. “I’d like one of the pastries in the leftmost row at the counter.”

“And I,” added Tim, “would like a piece of that brown cake with red berries on top.”

“For me,” stated Ronald, “one of the big cookies would be fine.”

“OK” said Liz “and for me I’ll have two natas.”

Before she could stand up, Jill asked “What are ‘natas’?”

“Natas are tiny Portuguese custard tarts,” answered Liz on leaving. “Absolutely delicious!”

When she came back she didn’t have only the specified desserts, but also a nice cup of their favourite coffee for everyone.

While eating, Liz mused, “Do you think that if I went to the dessert counter tomorrow and asked for exactly the same things, I would also get exactly the same desserts?”

Ronald reflected a bit and answered, “Well, actually they don’t have the big cookies every time.”

And Jill added, “and probably tomorrow they would arrange the pastries differently and mine wouldn’t be on the leftmost row.”

Tim continued, “I think I would still get a brown cake with red berries on top, but it would not necessarily be the same as today! Probably you, with your natas, are the only one that would get the same dessert!”

“You know what, people?” announced Liz, “You have just described exactly why you always have problems with the automation tool not recognizing the GUI objects it is supposed to drive!”

Now, all three of them looked really stunned. Ronald spoke for them all: “I don’t know what you’re talking about; please explain what you mean.”

Liz replied, “I noticed that you left it to the tool to define how to recognize the GUI-Objects, and so sometimes the criteria is the pixel position - the leftmost row -, sometimes the contents of the text field itself - brown cake with red berries -, at times other run-time properties like indices - big cookies - and finally sometimes the internal names used by the developers - my natas -. If you don’t define the criteria yourself, you can get in trouble again and again!”

“Hmm,” admitted Ronald, “we never even thought about this. I remember vaguely that they told us something about an object repository in the training, but it didn’t seem to be necessary when we recorded the test cases.”

“I suspected as much,” said Liz, “when I examined what you had already automated, and I have begun to do something about it.”

“Is there a pattern for this too?” asked Jill.

“Yep” answered Liz, “here it is” and shoved the tablet over the table so that they could see better (we have already seen it, but here it is again).

OBJECT MAP (Execution Pattern)

Declare all the GUI-Objects in the Object Map of the test automation tool.
(Or define your own if appropriate).

Context

This pattern is appropriate if you want your test automation to be long lasting, because your scripts will be easier to read and less dependent on the current tool.

This pattern isn’t needed if you only write disposable scripts or if your application is exceptionally stable, with object names never or hardly ever being changed.

Description

Declare all the GUI-Objects in the Object Map of the tool (note: this may also be called a GUI map or Object Repository). You can then write your scripts using standardized names. The scripts will be more readable and if you have to migrate to another tool you will not need to change the scripts to allow for new names even if tools seem never to use compatible naming conventions.

Implementation

This is one way to implement TOOL INDEPENDENCE, as your tests will be one layer removed from the specific names in the software.

Some suggestions:

- Ask the developers to use unique names to define the GUI-Objects. If the same objects are used over a number of different screens (object-oriented inheritance) it may be useful to add the screen name to the object name for better recognition.
- When driving the GUI avoid using pixels, screen coordinates or bitmaps to identify graphical objects. Screen coordinates can vary from machine to machine, from operating system to operating system or from browser to browser. Also changes in the Software under Test (SUT) can mean different positions for the GUI-Objects, so your scripts are more likely to break.
- If the SUT runs in different countries, you must also avoid using text labels if you don't want to write new scripts or have to have a separate Object Map for every new language!
- If you can't use the tool's facilities, you can make your own "translation table", which you have the names you will use in the tests in one column, and the current name used by the software in another column. Then if the software names change, you only have to update your translation table and all your tests will continue to work.

Some suggestions for object naming:

- Internal developer name: if the developers follow a style guide it can be useful to use the same names. In this way you have also the advantage to be able to communicate with the developers in their own "language".
- Label on screen: using the labels on the screen makes it easier for testers to follow what happens in the script.
- Functional term: using the correct terms is a useful way to document what the script is supposed to achieve.

Potential problems

This approach might be perceived as an unnecessary additional complication, but it is needed to gain a level of abstraction from the tool.

"In our dessert case," Liz added, "I would even refrain from using the internal name, natas, because it didn't resonate with any of you. I would call this object something like 'Portuguese-custard-tart', so that when one reads the script it would be immediately clear what it's about."

"And you have already begun to change the names?" asked Ronald. "Can you show us how it works? Then, when we have to update something, we could also build this OBJECT MAP."

"Of course," replied Liz "I'm doing it exactly like you say! Let's go back to work and I'll show you."

Back at work it was quickly done, and the testers promised to add to the OBJECT MAP every time that they had to control one of their scripts.

Liz thought to herself "Awesome! Getting dessert worked just like magic!"

1.1.6. Tim's accident

DATA-DRIVEN Test
finds a big bad vicious bug!
Bring on the Champagne!

A couple of days later Tim didn't show up for work and his wife called to inform the team that he had had an accident - he had been knocked off his bike by a car and was now in the hospital with a broken leg. The doctors could not say how long it would take for him to be able to return to the office.

"Oh no," groaned Ronald, "The automated tests were run last night. Now who is going to go through his test cases?"

"Ron," interrupted Jill, "I think he would rather do them himself instead of lying in a hospital! Don't be so rude!" She added, "We will just have to share them between ourselves." And addressing Liz, said "Can you do his automated test cases?"

"Sure" Liz answered "no problem, which ones?"

Later that morning, she was examining Tim's results from the latest automated test run and noticed with pleasure that a big set of the test cases had passed. "I really love green," she exclaimed and then mused, "but I wonder if they always pass?"

Turning to Ronald she asked, "Can I see the older logs? I'd like to see how these test cases executed in the past."

"Yea, sure, I'll send you the link" replied Ronald.

Liz studied the trend and noticed that the test cases that were always OK were all testing the same algorithm. Suddenly she recalled what Leroy had told her about all the unit tests the developers were writing and running. "Could it be," she wondered, "that these tests always pass because they have already passed the unit tests?"

Jill lifted her head. "Do you mean that we did all this work for nothing?" and added, "I hope that's not true!"

"Well," said Liz, "there is a way to find out: I'll take a look at the unit tests, it's something I wanted to do for some time anyway."

No sooner said than done! Liz was already browsing through the unit tests. And it wasn't long before she exclaimed "**Shucks! We can throw all these tests away.**"

Jill and Ronald came over to her desk. They didn't want to believe it, but she could show them all the tests that the developers had written, and they really tested the same stuff.

"Tim spent so much time getting these tests right," moaned Jill. "I can't believe it!"

Liz warily asked them, "Who decides what to test and what to automate? Don't you coordinate your work with the developers?"

Both Jill and Ronald looked abashed “We didn’t really believe that the developers were testing it too,” Jill said, “and no, after hearing from Leroy again and again that they didn’t have time, we just didn’t ask any more.”

“OK” decided Liz “I’ll go through all the unit tests and check if there are more duplicates. It’s a pity that you worked in vain, but, see it as positive that at least you won’t have to maintain these tests any longer!”

“You know what,” she added, “I remember having seen a pattern that could help us in this situation. I will try to find it and see what it suggests.”

So instead of going out to eat with the others Liz opened the wiki and started her search. “Ah,” she sighed after a while, “here it is: WHOLE TEAM APPROACH” and started browsing through it

WHOLE TEAM APPROACH (Process Pattern)⁴

Description

Everyone collaborates to do test automation. Developers build unit test automation along with production code. Testers know what tests to specify, automators or coders help to write maintainable automated tests. Other roles on the team also contribute, e.g., DBAs, system administrators.

Implementation

If you are doing agile development, you should already have a whole-team approach in place for software development, testing and test automation. If you are not doing agile, it is still very helpful to get a team together from a number of disciplines to work on the automation. In this way you will get the benefit of a wider pool of knowledge (SHARE INFORMATION) which will make the automation better, and you will also have people from different areas of the organisation who understand the automation.

Potential problems

If people are not working on the automation as a FULL TIME JOB, there may be problems as other priorities may take their time away from the automation effort.

Another possible problem occurs when many DevOps teams develop test automation independently (look up the issue LOCALISED REGIMES).

“Hmm,” Liz thought, “the developers don’t really consider us as belonging to the same team. We must find a way to show them our value! It would be great if we could use our automation to discover some really bad bug!”

When the others came back from lunch, Liz showed them what she had found and said, “We must find a way to demonstrate to the developers - and of course Tom & Jerry - that our automation brings real value to them.” She concluded rather disappointed, “but I have no idea how to do it. What about you guys?”

They nodded but gave no answer and so everybody went back to work.

⁴ Suggested by Lisa Crispin

Later in the afternoon, Ronald came to Liz's desk and asked, "Have you already removed Tim's duplicate test cases?"

"No, not yet," answered Liz, "Why?"

Ronald looked thoughtful. "I'm not sure, but maybe I have an idea about how to impress our developers."

Liz turned immediately alert and even Jill, who had her desk nearby, looked up.

"You showed us this morning," Ronald began, "that Tim's test cases are the same as the unit tests. Are they really identical? I know for a fact that there are hundreds of possible combinations that we should have tested, but it was just too much work and so we selected only a subset." He went on, his voice firmer, "If the developers chose the same subset then we really have tested the same thing, but," and he paused for a moment, "if we could show that we can actually test all combinations then I think we could..."

Liz was getting all excited and didn't even let him finish "Gotcha!" she exclaimed "That's just it!" and continued, "I know exactly how to do that. Ever heard of DATA-DRIVEN TESTING?" while she was still talking she was already browsing through the wiki. "Here it is" she said and showed them:

DATA-DRIVEN TESTING (Design Pattern)

Description

Write the test cases as scripts that read their data from external files. In this way you have only one script to drive the tests but by changing the data you can create any number of test cases. The charm is that if you have to update the script because of some change in the Software Under Test (SUT), you frequently don't have to change your data, so you don't have to spend too much effort in maintenance.

Implementation

Write a script with variables whose content is read sequentially from a file such as a spreadsheet. Every line in the file delivers the data for a different test case.

An easy way to implement this pattern is to use CAPTURE-REPLAY to capture the test initially. The captured test will have constant data (i.e. specific test inputs for every field). You can then replace these constants with a call to data that is read from an external file.

Potential problems

If your data is not contained in only one data file, you must make sure that the script and data are correctly matched.

Jill, who had come over to see too, clapped her hands "That's great!" she said "That's exactly what we need! How come we didn't think of such a simple solution?"

"And to change Tim's test cases accordingly is just a piece of cake!" beamed Liz. "I'll do that, and you guys provide the data."

Ronald nodded. "I have to talk to Paul tomorrow anyhow. I'll ask him what he needs." He went on, "He wasn't very pleased when we decided to do only a subset."

They went back to work, but nobody was really productive that afternoon.

The next day Liz had already started to turn one of Tim's test cases into data-driven, and she had even remembered to do the pattern OBJECT MAP and had therefore registered all the relevant objects in the tool. To complete it, she only needed some more information on the necessary parameters.

That afternoon, while they were waiting for Ronald to come back from his meeting with Paul, Jill moved over to Liz's desk and started to say, "I'm still stunned that none of us thought about something as simple as DATA-DRIVEN TESTING when we implemented those algorithm test cases. Are we really that stupid?"

Liz laughed "No, you're not; that happens to a lot of people. Didn't the developers fall into the same trap too?"

"You're right" agreed Jill "but still..."

"The problem," explained Liz "is that test automation is not only different from testing, but it's also different from development."

"Pardon?" interrupted Jill, "I thought that our problem was only that we're not good enough developers."

"Don't get me wrong," replied Liz, "having a developer mind-set and knowing how to code definitely helps, but that's not enough, otherwise the developers would have implemented data-driven scripts in their own unit tests themselves."

"I have recognized the difference myself only recently," she added. "You see, for a developer the main incentive is to implement something new as efficiently as possible - by the way, that's where GOOD PROGRAMMING PRACTICES come from - but in the case of test automation, to automate a new test case efficiently is not enough!"

"Why not?" asked Jill, looking puzzled.

"Well" answered Liz "think a moment about the why of test automation. Why are we willing to go through all this trouble in order to automate some test cases?"

"Isn't it obvious?" answered Jill rather piqued, "we don't want to waste so much time with those boring regression tests."

"Sure," continued Liz, "but why do we have to repeat the regression tests over and over again?"

"Now, you are really asking stupid questions, Liz," replied Jill pouting, "we must make sure that the developers didn't insert any unexpected and unwanted side-effects when implementing their new stuff."

"Exactly," said Liz and went on, "and that means that we must not only automate the tests but do it in a way that enables us to maintain them with as little effort as possible. Do you understand why now?"

Jill was somewhat puzzled but started to think it over. "I don't get it," she said finally, "the apps also have to be changed, so what's the difference?"

"Ok," answered Liz, "let me explain: with development, you may have to go back to old code to change it, so it's a good idea to make the code easy to maintain. But in test automation you are 100% sure that that you will have to run tests again, otherwise you wouldn't need automation at all!" She continued, "This means that your focus should not be only on implementing new stuff, i.e. automating new test cases, but, even more importantly, on doing it in a way that allows you to update it later as painlessly as possible!"

"When you automate something," she went on, "you must constantly think: If I have to change this script or this keyword six months from now, how can I make life easier for myself or maybe for some colleague? And that means that you have to use all the tricks of the trade from the very beginning."

Jill didn't look convinced: "I still don't understand! Do you mean that in regression testing you always have to test the old existing code as well as the new stuff?"

Liz tried to interrupt her, "Yes, that's true, but it's not the whole story. Let me try to explain it another way. Let's see: you probably don't know that I play the string bass in a jazz-band. In jazz you have pieces where everybody plays together and parts where each player gets to do a solo improvisation. Developing software is like when you are playing solo. In a concert you have a guideline, for instance a specific key, but you can improvise freely because you are playing alone. In software development you have specs to follow, but you are free to adapt them for technical reasons, for instance because a component that you had planned to use just doesn't work as expected. As long as you stick to the specified interface, the way you call a function, inside the function itself you are free to do as you like. Automation is like the band playing together. The band members must follow the score precisely otherwise you would get a 'cats' concert'. In automation the team must also follow the score, which in their case is the system under test, the so-called SUT, and must therefore be able to adapt quickly to any changes in the SUT."

Jill was thoughtful. "Yes, now I see what you mean. Can one learn to do it?"

Liz laughed. "Of course! Do you think that playing in a band just happens? Do you think that experienced automators are just born that way? It is often a painful journey and I'm impressed with the patterns in the wiki. By applying them from the beginning people can avoid a lot of trouble."

They were still talking when Ronald finally stepped in and he was beaming. "Paul was really taken with our idea," he said. "He will provide us with a list of all the combinations starting next week." He added smiling, "We are lucky to come up with this idea right now. Tom had just told him that lately some users had complained

about strange effects and nobody could pinpoint the exact causes, but they seem to be due to data combinations that weren't tested."

When she went home that evening Liz was thinking, "Now I really have a lot to tell Jack! After all it was good that they come this week-end and not earlier, or I wouldn't have had any good news to relate. While the boys will be playing with their new trains, we will be able to 'talk shop'. This must be celebrated! I must make something really special."

She went to her usual supermarket and looked over the meat, the fish and the produce before finally deciding on the menu she wanted to offer, eggplant parmigiana⁵. She had thought "parmigiana is very similar to lasagne, only instead of pasta there are eggplants, so the kids should like it and I know that Deirdre doesn't make it very often because it's so much work. As a starter, I'll make my special salad and for dessert we'll have ice cream with my home-made toffee sauce."

Lunch that Sunday was a complete success and Jack had really enjoyed hearing how she had explained the patterns to her colleagues. "I thought the patterns useful but still quite prosaic, but you make them sound like fun!" he had said chuckling.

Tuesday the next week, Paul handed over the list with all the combinations he thought should be tested. In the meantime, Liz had removed all of Tim's algorithm test cases except for the one she had reworked. Ronald and Jill prepared a file with all the combinations. On Wednesday everything was ready. The tests as usual started automatically in the evening.

The next morning, they were all at their desks much earlier than usual.

They were most interested in the new data-driven tests and yes, there were quite a few failures. "Gotcha!" exulted Liz. "These combinations have never been tested before and look how many bugs came up."

With a twinkle in her eye Jill said, "Now the developers will hate us and love us!"

"Yes," continued Ronald, "on the one hand they hate it when we find bugs, but in the long run they are grateful when we find them and not our customers!"

"And who is going to tell them?" asked Jill, hoping that the answer would not be: You.

"I'll do it," offered Ronald, "I'm looking forward to seeing their faces."

In the next week there was a real frenzy of new builds and for each one, they launched the data-driven tests, and still every time they continued to show at least

⁵ See attachments for recipes

some red. They could definitely sense the developers' frustration. Finally, with the Thursday night tests came the long-awaited result: on Friday all the tests showed green!

That afternoon, suddenly the testers and Liz were called to the big meeting room. Still wondering what was going on, they entered the room and were greeted with a clapping ovation. Paul shook hands with each of them and led them to the centre of the room. There they were greeted by a man that Liz hadn't seen before. Jill nudged Liz with her elbow and whispered, "That's Jerry!"

Jerry handed each of them a champagne glass and then the whole room were raising their glasses and cheering.

Trying to sound solemn, Jerry said "I want to thank you all for finding a really vicious bug! Thanks testers and test automation team for your really comprehensive tests!" and saying so he raised his glass toward Ronald, the other testers and Liz. Then he continued, "Thanks to all developers for checking every line of code and thus demonstrating that the bug resided in my very own mathematical algorithm!" This time he pointed at the developers and tried to scowl, but then had to laugh. Finally, he concluded "Without your help I would never have been able to find this bug and in the long run it would have caused us more and more trouble. Thanks again everybody! Please enjoy the buffet! I wish you all a wonderful week-end!" And he showed the way to a rich buffet table.

After clapping some more, the noisy mass stormed the table and for a time everybody was busy eating.

Later, standing at a tall table, Liz and the other testers were discussing the happening when Paul dropped by. He asked them "Well? How do you like your party?"

Ronald answered first: "I'm speechless! I would never have imagined that we would get so much recognition and appreciation."

And Jill continued, "and in public too."

"Well" answered Paul, "a good management principle is to recognize and celebrate success. Nothing motivates more!"

At the words 'celebrate success' something clicked in Liz's memory and she exclaimed "Why, this is already the second time that you are suggesting or applying test automation patterns! First it was to DO A PILOT and now CELEBRATE SUCCESS!"

Paul laughed, "I was a senior manager at a management consultancy. I'm expected to know what I'm talking about if it involves management!" and added, "It seems that your patterns are right on target! You should show them to me sometime."

"Now come," he said then to Liz, "Since he has not been presented to you yet, Jerry wants to thank you personally."

Liz had been very favourably impressed by Jerry's frank admission that the bug had been his creature and so was quite curious to meet him.

Paul led her to another table where Jerry and Leroy were standing and said, "Jerry, here's our new automator, Liz."

"Welcome Liz, thank you" Jerry stammered, his ears turning red like tomatoes on seeing her. After a short while to collect himself, he continued, "I want to thank you again for your contribution and I feel bad for not having introduced you to the technical features of the apps yet, but I just didn't have time."

"Thank you, Sir," replied Liz smiling warmly, "I have been going through lots of unit tests lately and I think I've got a good idea of how the apps work."

Jerry was surprised. "I can't imagine that going through the unit tests can give you more than some very local information! You will have seen the leaves but not the forest!"

Liz smiled at the comparison and answered, "Yes, you are right, but still I consider unit tests to be the most efficient way to document a program."

Jerry actually agreed but was interested to hear her reasoning. "Better than specifications, use cases etc.?" he countered.

"Well," she replied, "Specifications, use cases and so on are of course needed in order to be able to know what functionality to implement, but later, when a function or a method evolves and is updated again and again, only the unit tests offer a consistent and up-to-date documentation. I started out as a developer, so I know what I'm talking about. In the beginning, you write a lot of comments, describing the input parameters and the results. At the next change maybe, the documentation will also be updated, but usually one is a bit stressed and defers it to 'later'. After some time this kind of documentation has usually become utterly useless. But," and here she looked at him earnestly, "this is not the case with unit tests. If they are no longer needed then they are removed, but otherwise they are always up-to-date, or they wouldn't pass!"

"Yes, you're right," agreed Jerry, feeling more and more at ease with her and went on, "but I'd still like to give you an overview, and since you already have a general idea, I'll only need a couple of days instead of a whole week. What about next week? Monday? And, please call me Jerry!"

Liz looked surprised at being asked and said, "Of course, you're the boss!" adding uncertainly "Jerry."

Leroy had listened attentively and now asked, seemingly upset, "Liz, why are you looking at our unit tests?"

Liz answered, "We discovered that some of the tests automated by the test team actually tested exactly the same things as some of the unit tests, and we want to make sure there's no more duplication. If the unit tests have passed, there is no need to test the same stuff in the same way, although the end-to-end tests may still cover some of the same ground."

"What a stupid thing to do!" exclaimed Leroy. "Why didn't you just ask us before automating those tests?"

Liz didn't really want to discuss it in front of management, but on the other hand she was not one to just duck under. "It was before my time," she answered innocently, "so I don't really know, but I suspect that the developers were much too busy to waste their time with the testers..."

She hadn't finished her sentence when Paul sputtered out laughing behind his hand. Leroy turned tomato red. Jerry was able to keep a straight face and had the presence of mind to reply "A propos busy, we still have to solve that other bug. I'd like to work on it this week-end, but I still need some input from you," and he nudged Leroy out of the room.

Paul was now openly laughing and said, "I notice that you too have already got to know Leroy! Never mind him. He's like that with everybody. I don't think he really means it. Jerry is one of the few people who knows how to handle him."

Liz was still a bit dazed, but finally she answered, "Awesome! With Jerry he was as meek as a lamb!"

Paul nodded "Come closer," he said, "I'll tell you a story. When Tom met Jerry the first time in college everybody started teasing them with their names, Tom & Jerry. Now, Tom has always been confident and sure of himself, so he didn't care, but Jerry was really hurt. He was a brilliant student, but he had already been mobbed in school because of his big ears and still had no friends and was almost pathologically shy. On a whim Tom decided to do something about it. He ordered t-shirts for Jerry and himself, one with Tom the cat and one with Jerry the mouse and convinced Jerry to wear it with him."

Now it was Liz's turn to giggle. "And it worked?" she asked.

"Like a charm!" answered Paul with a wide grin and continued "When he was wearing that t-shirt he really became Jerry the mouse and was suddenly naughty and played all kinds of mischief on his supposed tormentors. He became great friends with Tom and in time acquired the confidence that you now see." These last words he said with a twinkle in his eyes, clearly remembering Jerry's reaction on seeing Liz. Finally, he finished, "And that's the reason why he can work so easily with Leroy: he recognizes him as very similar to his own old self!"

"Awesome!" exclaimed Liz and asked, "Is Jerry also musical? I mean, he should be since he was able to create the Screech-apps."

"Well" Paul answered, "I believe he is the least musical person I have ever known!" After a short pause he went on "Don't get me wrong: he loves music, especially Bach, but he doesn't understand it."

And as Liz, completely surprised, just stared back, he continued, "I asked him once myself about it and he said that probably this was the main reason why he could actually write the music software! He said that not knowing or understanding music

made him more sensitive to the statistical patterns that distinguish one musical genre from another!"

"Awesome," repeated Liz, "I would never have guessed that!"

1.1.7. Testing Know-how

The developers
and equivalence classes:
evil in your eye!

On Monday Liz was really keen to hear what Jerry would tell her about the apps. She didn't have to wait long. When she went to get her first coffee she found Jerry there with a cup in his hand.

"Fine," he said, "We can start right away. Let's go into the small conference room."

Liz followed him. In the room there were already two other new employees, Rich (Development) and Sheryl (Marketing).

That first day Jerry explained in general how his algorithm worked, but didn't really go very deep into technical details, so that even Sheryl could easily follow his explanations. He also spent some time describing what he wanted to implement next, and Liz noticed with pleasure that some of the topics had been proposals that she had discussed with Paul in her introduction week.

On Tuesday only Liz and Rich were left and Jerry became much more technical. At the end of the day he asked Liz if she needed more details. He had planned another two days with Rich and she could just join them.

Liz thanked him but answered "I think I have a pretty good idea now. But may I come to ask you if I do have some extra questions?"

"Of course," replied Jerry slightly disappointed and so she returned to her desk, picked up her stuff and was done for the day.

On Wednesday morning Liz was again browsing the unit tests to see if there were more duplicates or even candidates for data-driven testing, and the other testers had fallen back into their usual routines of updating old automated tests and running manual tests.

Then she found some tests that seemed really similar, but she wasn't sure and asked Ronald for help. "Ron," she said, "can you please look at these unit tests? They may be candidates for data-driven, but somehow they're different, I'd like to hear your opinion."

Ron came over and took a look. Then he exclaimed "Wow! What did you say last time, Liz? Gotcha?" and continued, "These tests all test the same equivalence class and therefore all but one are completely superfluous. This shows that our dear developers have no idea of test design." And he concluded, "You, Liz, wanted some more 'ammunition' to convince them to collaborate with us: this is it!"

Liz looked perplexed, "Hmm, what's an equivalence class?"

Ron and Jill looked at each other. "You've never been a tester, right?" said Ronald. "Only developers are known to have no idea."

Liz nodded, "Yes, I was a developer before I turned to test automation, so what?" and added, a little piqued, "You still didn't answer my question."

Ronald had an evil twinkle in his eyes. "Yes, I will explain it to you, but not here. I have an idea," and he asked her, "Please show me the list of the test cases you're supposed to automate."

Liz nodded, but one could see how doubtful she looked. "It's the file you gave me some time ago" she answered, "Got it? Or should I send it to you again?"

Ronald went back to his desk, opened the file and started to read. Jill looked sympathetically at Liz and said, "I know that look in Ron. He's a sly one. I wonder what he's scheming."

The women went back to work and it seemed a very long time before Ronald got up and signalled for them to come over to his desk.

"I've found just the right thing," he started and continued, "It's almost lunch time. We'll go sit next to some developers, apparently by chance, and you, Liz, will ask me about these test cases. I'll manage the rest," and he explained to her exactly what she was supposed to say.

Lunch time came, and Ronald said, "Let's go to the cafeteria at Software Forge again." And added with a wicked grin, "This time I will need some hamburgers."

They went in and all but Jill got hamburgers - her choice today was pizza margherita. Then they took a 'casual' look to see if there were already some of their developers. They were lucky, they saw Tabisha, Jim and Leroy sitting together at a big table and there was still enough room there for all three of them. They went over to the developers. "Are these seats taken?" Ronald asked, and when they shook their heads, he called to Liz and Jill to come over.

After a short while, as planned, Liz said to Ronald: "I have these test cases to automate next," and briefly described them to him. "I looked up the tests for this part of the app, but I have the impression that there should be more of them than there are. Are you sure that so few are enough?"

"Of course," he asserted, "I have located the equivalence classes and you don't need more than one test case per class."

Liz looked sceptical, "Hmm, what's an equivalence class?" she asked.

Ron and Jill looked at each other. "You have never been a tester, right?" said Ronald, "I notice that developers often don't know about this."

Now the developers that were listening on the other side lifted their heads, and Leroy glowered. Ronald grinned inwardly.

Liz nodded, "Yes, I was a developer before I turned to test automation, so what?" and added, "You still didn't answer my question."

"You're right," said Ronald apologetically, "I'll explain it." He looked thoughtful for a moment. Then he went on "See this hamburger? How would you test it?"

Liz looked puzzled. "A hamburger? What would I want to test there?"

Ronald was unruffled. "Well, for instance if the meat in the patty is really beef or the salad is fresh etc."

Liz replied, "I see. I would take some samples of the meat, of the salad, the bun and so on and check each of them for what they are supposed to be."

Ronald agreed "Yes, those would be your test cases. But now consider that you want to do the testing as efficiently as possible. How many samples - test cases - would you need?"

"Hmm," I think I begin to understand, she said, and added, "If the meat in the patty is homogeneous I will only need one sample for the meat. The same for the bun and for each ingredient in the salad."

"You got it," Ronald answered, "and these are exactly the equivalence classes I was talking about." He continued, "To better understand I'll make another example."

Now the developers had stopped talking to each other and were listening, while still pretending not to be interested.

"Do you know how a driver's licence works in California?" he asked and took a napkin and drew something like this:

Type	Learner's permit	Restricted Licence	Full Licence
Min. Age	15 years, 6 months	16 years	17 years

Figure 1.1.7-1

"Now," he added, "what are the equivalence classes in this case?" and nodded to Liz, "What do you think?"

She considered a bit and then took another napkin and drew:

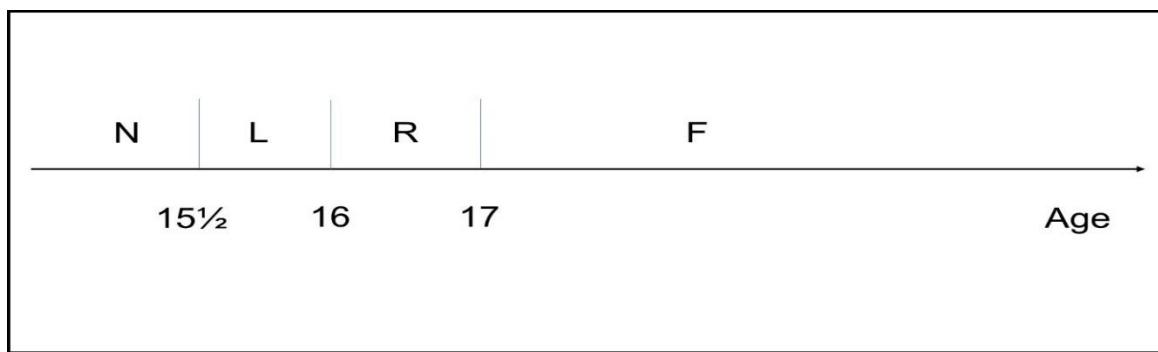


Figure 1.1.7-2 Equivalence classes

Then she said, “N is the class where a driver’s licence is not allowed, L is where you can get the learner’s permit, R the restricted one and finally F the full permit. Did I get it right?”

“Great,” beamed Ronald, well knowing that there was more that could be said about equivalence classes and boundaries, and added, “and now you see why you need only one test case for each class: even if you do more you will always get the same answer, pass or fail, because you are always just testing the same code and so it would be only a waste of time!”

Tabisha had been leaning over to see better. Now she said, “That was very interesting. In fact, I had never heard about equivalence classes either, and I think I can use this for my unit tests,” and asked, “Are there other such tricks?”

Jill started coughing and Liz realized that she was desperately trying not to laugh.

Ronald kept cool and replied, “Of course! Are you interested? There is more about equivalence classes for a start, but also lots of other ideas. Today is too late, but we can meet at lunch again and I can explain about more of these ‘tricks’.”

Tabisha and Jim nodded, and Jim said, “Yeah - That’s cool, man!” Leroy said nothing, but he had been listening attentively. In fact, he was wondering whether the testers might find fewer bugs in his code if he secretly applied this technique. (But he was not about to admit to that, especially in front of any testers or worse still a cat-lover like Ronald!)

On the way back to work testers and developers chatted together as the best of friends, but when they reached their office, Ronald, Jill and Liz started to laugh like crazy.

Finally, Ronald said, wiping the tears from his eyes, “See, in this way we haven’t shamed them for not knowing about equivalence classes and instead made friends. Will this help us implement that pattern that you showed us a few days ago? What was it called again?”

“I remember,” interrupted Jill, “WHOLE TEAM APPROACH, right?”

Liz nodded with a smile at Jill and pointing a finger at Ronald said, “Now I understand what Jill meant when she said you were a sly one!”

1.1.8. Jerry comes on-board

The pattern tells all
LAZY AUTOMATOR is
the mind-set for you!

The next day it was almost a crowd going to lunch together because a few more developers had joined: Chris, Alice and even Jerry.

At the table, Ronald and Jill explained the equivalence classes to the developers that were missing the day before and introduced the concept of boundary-value testing. Tabisha said, "Hmm, that's very interesting - it reminds me a lot of some of the most frequent bug reports I get from you testers! Now I know your secret!"

When the developers asked what would be the topic next time, Liz broke in and said "Next time it will be test automation patterns time!"

"Oh good," exclaimed Chris, "I've always wanted to use patterns in my unit tests!"

"No, no," replied Liz "system test automation patterns, not unit test patterns."

"Huh? And what's that?" asked Chris, somewhat annoyed.

Liz explained, "The patterns I want to talk about refer to system test automation, but many of them are also useful for unit tests. Why don't you just come and see?"

The developers grumbled among themselves. Liz thought "I'm really curious to see if some of them do show up tomorrow."

The next day only one developer came, and surprisingly, it was Jerry. "I want to understand what you're up to," he said smiling at Liz, and added, his ears an incredible red tinge, "I always like to know what are the latest innovations and I had never heard of these patterns before."

Liz was at the same time both flattered and fluttered. She had intended to talk about LAZY AUTOMATOR, but now was considering maybe to do SET CLEAR GOALS instead; after all, Jerry was management. But after a brief thought she decided to stick to her original plan. Jerry was management, but he was first a developer and she wanted to 'attack' him there.

This time Jill suggested the cafeteria of Artartica, an online marketplace for self-made products and art. Here the food was organic and often vegetarian or even vegan.

At the table Liz began with: "A couple of days ago, Jill and I were discussing how developers and test automators have, or should have, a different focus to their work. Later, at home, as I was reading through the patterns, I happened to find one that exactly describes my point of view, LAZY AUTOMATOR."

"Now that's a good name for a pattern!" interrupted Ronald with a broad grin.

"I hope that it doesn't really mean what the name says," added Jerry with an awkward smirk, "or I'll have to fire all my automators."

Liz smiled innocently and continued, opening the appropriate page on her tablet: "Here it is. Before you sneer, have a look at it"

LAZY AUTOMATOR (Process Pattern)

Description

Why are lazy people the best automation engineers? They are too "lazy" to do the same boring job over and over again. Instead they devise ways to shortcut it with help from scripts or tools and profit from common good practices. Whenever they find themselves (or someone else) repeating a task, especially if it is "mechanical" in nature, it could probably be automated (provided it doesn't take too much effort).

For a "lazy automator" the focus is not only on building efficient testware but also on always keeping in mind that this testware has to be maintainable. Will he or she be able to update this testware six months from now with only a minimum of effort?

Implementation

Since what you have to automate varies from application to application, there can be no general solution; what is important is to be aware that there usually is a more efficient way to do something.

The following patterns are crucial in making your automation process and your automation testware efficient:

- **ASK FOR HELP**: don't waste time trying to do everything by yourself. If somebody knows better, do not hesitate to ask for help.
- **DON'T REINVENT THE WHEEL**: if a problem has already been solved, then make the solution yours and go on from there.
- **THINK OUT-OF-THE-BOX**: The best automation solutions are often the most creative.
- Design **MAINTAINABLE TESTWARE**, which means that you should adopt **GOOD PROGRAMMING PRACTICES** and a **GOOD DEVELOPMENT PROCESS**.
- See that you can draw on **DEDICATED RESOURCES**: if the test automation team members are jumping from one project to another, they will not be as effective as they could be. A lot of time will be wasted, and not only for the test automation project, just trying to remember where they stopped and what comes next. Also if you don't have exclusive use of the machines you need, it may be much more difficult to get the initial conditions right, because other developers or testers will corrupt your data over and over again.
- **LOOK FOR TROUBLE**: Keep your eyes open for possible problems; the sooner you spot them, the easier it will be to solve them.
- Try to adopt **TOOL INDEPENDENCE**: sooner or later your SUT will change in a way that renders your current test tool obsolete. Or your tool will not be supported any longer and the next tool, even if developed by the same vendor, will not be compatible with the old one. If you keep your dependencies on the tools to a minimum, you

will be able to face such events without having to start again from scratch.

Potential problems

We say that the automator should be "lazy", but only in the sense of not wanting to repeat actions that could be automated.

Sometimes in trying to achieve that goal, automators might become so fixated on trying to automate something, that they don't realise how much time they have spent on it, and this can be wasteful. It is only a net gain if it can be automated at a reasonable cost (in effort and time).

After reading it, Jerry exclaimed, "Why, this sounds quite reasonable and I think it could also be useful for our developers!"

And he dared to ask Liz jokingly "Are you a LAZY AUTOMATOR, Lizzy?"

Liz was not ashamed to confirm that she was and asked slyly "Are you going to fire me now?"

All laughed, and Jerry replied, his ears burning again, "No, but maybe I'll consider calling you LAZY instead of Lizzy!"

Liz blushed and to give her time to compose herself, Jerry asked Ronald, "What are the words in capital letters here? Other patterns?"

"Yes" he replied, "the patterns are always written in capital letters in order to identify them."

"I see" said Jerry. "Liz, please show me one of the other patterns suggested here, for instance MAINTAINABLE TESTWARE."

Already Liz was on the right page:

MAINTAINABLE TESTWARE (Design Pattern)

Description

Identify the costliest and/or most frequent maintenance changes, and design your automation to cope with those changes with the least effort. When adjustments really are necessary, then they should be relatively easy to implement. For example, if objects are frequently renamed, construct a translation table from the name you want to use in the tests, and put in whatever the name of the object is for the current release of the SUT (OBJECT MAP).

Implementation

Some suggestions:

- There are many options to make and keep the testware maintainable, but to adopt a GOOD DEVELOPMENT PROCESS and GOOD PROGRAMMING PRACTICES is a very good bet: what works for software developers works for test automation just as well!

- For example, if your scripts are mainly “stand-alone” without much reuse, and automators are frequently re-inventing similar or even duplicated scripts or automated functions, then GOOD PROGRAMMING PRACTICES are needed, particularly DESIGN FOR REUSE and OBJECT MAP.
- Implement DOMAIN-DRIVEN TESTING: test automation works best as cooperation between testers and automation engineers. The testers know the SUT but are not necessarily adept in the test automation tools. The automation engineers know their tools and scripts but may not know how best to test the SUT. If the testers can develop a domain-specific language for themselves to use to write the automated test cases, the automation engineers can implement the tool support for it. In this way they will each be doing exactly what they do best. The advantage for the automation engineers is that in this way the testers will maintain the tests themselves leaving the engineers more time to refine the automation regime, and to solve technical maintenance problems.
- For example, if you have structured and reusable scripts, but there is a shortage of test automators to produce the automated tests (or they are short of time), this pattern gives the test-writing back to the testers, once the automators have constructed the infrastructure for the domain-based test construction. Other useful patterns are ABSTRACTION LEVELS and KEYWORD-DRIVEN TESTING.
- Don't forget to actually MAINTAIN THE TESTWARE: maintainability alone is not enough, it must actually *be* maintained.

Potential problems

Don't wait too long to build maintainable testware - this is best thought of right at the beginning of an automation effort (although it is also never too late to begin with improvements).

Now Ronald interrupted, “Well, yes that all looks good, but how do you actually do it - how do you MAINTAIN THE TESTWARE?” he asked.

Liz quickly showed the new page:

MAINTAIN THE TESTWARE (Process Pattern)

Description

Test automation scripts and test data need to be kept in step with the systems and software that are being tested; as the production software changes, it may cause scripts to fail or use the wrong data. If the testware is not regularly maintained from the very beginning, the automation will fall into disuse and eventually die.

The biggest factor contributing to excessive maintenance is the structure of the testware, but constant maintenance is still needed throughout the useful life of the automation.

Implementation

Here is some advice for keeping on top of testware maintenance:

- If you find an error in the test automation scripts, correct it as soon as possible.
- If tests are too complicated, simplify them (KEEP IT SIMPLE).
- If a test case doesn't pass any longer due to changes in the Software Under Test (SUT), check if it is still valuable and if it is, adjust it accordingly (scripts or data). If it isn't, KILL THE ZOMBIES.
- Verify occasionally that the returned results really are correct. After some time, FALSE PASS has a way of slouching in.
- REFACTOR THE TESTWARE, possibly at a time when code is not changing (you can't refactor tests and change code at the same time!)
- Run all your tests regularly. You will quickly notice what is obsolete and what is not.
- PLAN SUPPORT ACTIVITIES: Plan for regular maintenance as part of the ongoing test automation effort.
- Use DEFAULT DATA when your tests need a lot of data, but only a tiny part of it is really pertinent to the test case.

Potential problems

You may find that certain maintenance tasks are taking an increasing amount of time. In this case it may be time to set aside time for a major restructuring and REFACTOR THE TESTWARE. Refactoring is like a spurt of dedicated and concentrated maintenance but carried out so that future maintenance becomes easier after the restructuring.

It may be hard to justify the time taken to maintain the testware when deadlines are tight, and resources are in short supply. But without maintenance, your automation will die, just as a plant dies without water.

"Hmm," said Jerry, "this last pattern could be taken as-is for normal development too!" Liz smiled and said, "Exactly - refactoring is appropriate to development as well."

Then Jill asked Liz, "Show me that pattern KILL THE ZOMBIES - it sounds intriguing."

"Sure, no problem," replied Liz and went to the appropriate page.

KILL THE ZOMBIES

Implementation

Check regularly for test cases, scripts or data that aren't used any longer and remove them from the system.

Be suspicious if tests seem to be passing too easily. If you are "taking over" existing automated tests from someone else, make sure you know what all of the tests are actually doing.

Write some scripts or use monitoring tools to find out what is going on.

Potential problems

If your tool doesn't support you here, use the "find in files" functionality of a simple text editor to check what is actually still in use.

"Hmm," said Jill, kind of disappointed, "I was thinking of something more like lasers and aliens..."

Ronald laughed, "You and your fantasy books."

And Jerry added, "Better so. I wouldn't really care for aliens running around with lasers in our office." He went on, "but actually the pattern is sensible, and you should definitely apply it."

He was now quite interested and so he asked Liz, "Where do these patterns come from? Is there an official site?"

"Yes," she replied, "there is a wiki, where you can find all the ones collected as of today," and added, "and what's more important, there are also issues or problems that often arise with test automation, and each issue suggests the pattern or patterns to apply in order to solve it."

"And," Jill broke in, "there is a diagnostic feature that Tim originally showed us that helps find the issue that is most pressing at any time. In fact," she added, "we discovered that we needed expert help by using it." and pointed to Liz.

"I like that," said Jerry, "why don't we look at the Diagnostic?"

"Right now?" asked Ronald.

"No, I'll talk to Tom first, and I would also like to have your opinions on the current state of our automation. I'll come back to you in the next few days."

"Ah", thought Liz, "Maybe he too will find out that we need a TEST AUTOMATION FRAMEWORK!"

1.1.9. The Root of the Problems

Not a good approach,
TOOL-DRIVEN AUTOMATION:
Too hard to maintain

Going back to the office, Liz and the others were really abuzz. What would come out of this diagnostic with the managers? Liz hoped that Jerry would also find that a TEST AUTOMATION FRAMEWORK would be just the perfect solution!

Jill, Liz and Ronald considered what they were going to say when asked about the current state of automation.

"I think we should just stick to the facts," said Liz. "The old test cases were not automated very efficiently, but on the other hand, as you yourselves recognized, there were no automators and you could not have known better."

"You're right," said Jill, "but what if he just kicks us out?"

"We'll have to risk it," said Ronald. "Jerry is no novice and would immediately notice if we try to cover up. I'm with Liz for getting out the facts as they are."

The discussion stopped there and they all returned to work with their own thoughts.

That afternoon Jerry called Liz to his office. The others looked worried as she left. "I wonder why Jerry called her first," said Jill and added, "I'm afraid that he really will kick us out."

Ronald was not very convinced, but he thought that it was no time to let the head hang down. "Aw, don't worry Jill," he said reassuringly, "Jerry knows that we do good work, wait and you'll see. Remember what they said at that amazing party!"

Meanwhile Liz entered Jerry's office. "Close the door," said Jerry and added, "Please take a seat."

Liz complied and waited to hear what Jerry had to say.

"I called you first," Jerry began, "because I want to hear your opinion as, so to speak, an outsider." He went on, "Please be frank, I want to have a clear idea of what's going on before I talk to Tom."

Liz took a deep breath and began "The best way I can summarise the situation here is with an issue in the Test Automation Patterns wiki. Look up the issue *TOOL-DRIVEN AUTOMATION* and you'll have your answer."

Jerry wasn't sure where to find it, so after Liz told him that it was a Process Issue. Jerry clicked on "Process Issues" in the left-hand navigation, and then clicked on the last issue listed:

| *TOOL-DRIVEN AUTOMATION (Process Issue)*

Test cases are automated using “as is” the features of a test automation tool

Examples

In older tools this means using the capture functionality to develop test cases: while a test case is executed manually, the tool records all the tester actions in a proprietary script. By replaying the script, the test case can be executed again automatically (look up the pattern CAPTURE-REPLAY for more details).

In more modern tools you can create “keyword” scripts, but the mode of operation is actually much the same as in capture-replay.

Both approaches harbour serious problems:

The speed and ease in recording test cases can induce testers to record more and more tests (keyword scripts), without considering GOOD PROGRAMMING PRACTICES such as modularity, setting standards and so on that enhance reuse and thus maintainability. Without reuse, the effort to update the automation in parallel with the changes in the SUT (System under Test) becomes more and more grievous until you end up with STALLED AUTOMATION.

Resolving Patterns

Most recommended:

- TOOL INDEPENDENCE: Separate the technical implementation that is specific for the tool from the functional implementation.
- GOOD PROGRAMMING PRACTICES: Use the same good programming practices for test code as in software development for production code.
- LAZY AUTOMATOR: Lazy people are the best automation engineers.
- TEST AUTOMATION OWNER: Appoint an owner for the test automation effort.
- TESTWARE ARCHITECTURE: Design the structure of your testware so that your automators and testers can work as efficiently as possible.
- WHOLE TEAM APPROACH: Testers, coders and other roles work together on one team to develop test automation along with production code.

Other useful patterns:

- LOOK FOR TROUBLE: Keep an eye on possible problems in order to solve them before they become unmanageable.
- MAINTAINABLE TESTWARE: Design your testware so that it does not have to be updated for every little change in the Software Under Test (SUT).
- SET STANDARDS: Set and follow standards for the automation artefacts.
- GET TRAINING: Plan to get training for all those involved in the test automation project.

When Jerry lifted his head from reading, Liz said, "That's why yesterday I chose to talk about the pattern LAZY AUTOMATOR."

"I see" said Jerry. "It doesn't look good, right?" Then he asked, "What have you done up to now to improve the situation?"

Liz told him about her initial problems not getting support either from development or from testing and told him of her encounter with Tom. She remembered how irked she had felt when Tom had said,

'Just do your job, automate the missing test cases and let everyone else get on with their work. I know that test automation is valuable, and I support you 100%, but the daily work is much more important, that's where our bread and butter are coming from.'

"You did want me to be honest, didn't you?" said Liz.

Jerry looked thoughtful and then said, "I'm sorry that Tom wasn't very supportive, in spite of what he said. He had been rather stressed that week, but if he didn't want or couldn't give you the support you needed, he should at least have said so instead of promising to back you up 100%!"

"I must admit," Liz replied with a relieved smile, "that I had thought of telling him so myself and not in such kind words!"

Jerry chuckled and continued, "I'm glad you didn't - who knows how he would have reacted!" Turning serious he then asked, "And what did you do then?"

Liz described how she had started a small private pilot project to try to show people that she really could help and told him of her experiments with the core parameter keywords and how relieved Ronald had been after realizing that she had not really 'messed up' their test cases.

She told him how she had been able to show the other testers the advantages of applying patterns like INDEPENDENT TEST CASES and OBJECT MAP.

She then briefly mentioned the introduction of DATA-DRIVEN TESTING, and she still remembered very fondly the 'celebration party', and how impressed she had been at his reaction to them finding a bug in his code. Finally, she explained how the idea behind teaching test design to the developers had been to apply the pattern WHOLE TEAM APPROACH. Now she could rely on help from the testers and maybe soon also from at least some developers. She finished with a sly smile, "I still need support from management though."

Jerry had listened attentively and then asked, "What do you think the next steps should be now?"

She would have loved to just say TEST AUTOMATION FRAMEWORK but decided not to force the situation and said instead, "Well, I would go on with your plan to consult the diagnostics. The wiki collects so much experience, I would build on that."

Jerry had seen the flutter in her eyes and her slight hesitation and prodded her, “That’s not what you were really thinking, right?”

Liz blushed and admitted, “You’re very perceptive! I do have a pet pattern, TEST AUTOMATION FRAMEWORK, because I have already implemented one before and I believe it would be the right solution here too, but...”

Jerry interrupted her and said, “No buts. I told you to be frank with me. I’m glad to hear that you have made some good progress already, both in the new automated tests and in beginning to re-structure the existing automation. But I will also consider what the diagnostic suggests. Who knows,” he added with a smile, his ears a splendid red again “maybe it will be TEST AUTOMATION FRAMEWORK!”

Finally, checking himself, he said “Ok, thanks very much. I really appreciate your honesty.” Then he added, “Can you please ask Ronald to come in next?”

And as she went through the door, he thought that she was really the right person for the job. But hours later what he remembered was the twinkle in her eyes as she had admitted to being angry with Tom.

Liz on her side realised that she was beginning to like Jerry a little too much. “Careful, Liz” she told herself “this man is not for you!”

On returning to her office she told Ronald that Jerry was waiting for him.

As soon as Ronald was gone, Jill came over to Liz’s desk and asked, “How did it go? What did he ask you? What did you say?”

Liz smiled “Take it easy. One question at a time! He wanted to hear my opinion about the automation as an outsider and I told him that I thought the issue was *TOOL-DRIVEN AUTOMATION*.”

“Oh” said Jill “Tim had said that too when we tried the diagnostic. I’ll have to look it up,” and she asked again, “and then?”

“And then,” answered Liz, “I told him that it would be good to see what patterns the diagnostic suggests to move forwards from where we are now.”

“And then?” asked Jill again.

“And then he asked me to call Ron,” said Liz with a friendly smile “Don’t worry! Take it easy! Let’s wait and see what Ron will tell us.”

In the meantime, in Jerry’s office, Ronald was telling Jerry how easy it had been to start using AutomatePro and how he had just got carried away and had never stopped to think about maintainability. He even admitted that at first, he had been angry with Liz for “messing up his keywords”, but that he was very happy now with what Liz had accomplished in her time at the company.

Jerry laughed and said, “Well, it’s never too late to learn,” and continued, “I’m glad that Liz could show you and the others that it’s worthwhile to consider what you are

doing before just running on blindly." Then he added, "I will discuss the situation with Tom and Paul and will consider what the diagnostic suggests in order to find a way to move forward even more. I will probably call you and the others to help me select the patterns we should implement first."

When Ronald reached his office, he let out a relieved breath. Back at his desk he could reassure Jill and Liz that Jerry was looking for solutions and not for scapegoats and added: "He even told me that we will be helping him select the patterns to implement first!"

Liz was already day-dreaming about a TEST AUTOMATION FRAMEWORK and that Jerry was so cute with his red ears...

1.1.10. Tom & Jerry

Technical issue
not a management issue?
Maybe it is both

The next Monday, there was a very pleasant surprise: Tim was back, hobbling along with one leg in a plaster cast and a pair of crutches!

"How great to see you! How are you feeling?" Everyone crowded around Tim at his desk. He told them that he would only be doing half-days for the next few weeks. Jill immediately brought him his coffee and promised to keep him well provided.

After the initial excitement had calmed down, Tim asked how they had got on while he was off. Jill, Ronald and Liz looked at each other, wondering how to tell him what they had done with his test cases.

It was Liz who spoke first. "We have been doing quite a lot of improvements to the automation, and one of the first things we did was to transform your tests to DATA-DRIVEN TESTING. In this way we can test much more thoroughly than before."

Jill thought, "Why, that's really a good way to bring him back into things, by mentioning his contribution."

Eventually Tim had a good idea of what had been happening, and he was proud that it had been his test case that had found the famous bug. He was only sorry to have missed the party.

"So now our problem is to try and get Tom and Jerry to approve further progress?" asked Tim.

"That's about it," nodded Jill.

In the meantime, Jerry had been seeing Tom.

Jerry, never known for being particularly diplomatic, had started with "Our test automation stinks!"

Tom, who was expecting a completely different topic, was surprised and said, "I will have to fire Liz; she came complaining to me some time ago instead of just getting on with automating the new test cases."

"Yes, she told me about that," answered Jerry. "You really have a way to motivate people!"

As Tom looked startled, he continued, "You told her you support her 100%, but instead of actually doing so you more or less told her to just get going."

"Now, wait a minute," Tom countered, "You sound exactly like my Dad! I couldn't help her. Everybody was more than busy."

"Sure!" replied Jerry, "but then just tell her! The problem is not that you couldn't support her, but that you first loudly promised to and then didn't deliver! That's what 'really motivates' people!" he added sarcastically.

Tom considered for a while then said "Hmm, I hadn't considered it from that point of view. I thought that by saying that I support her, I could mitigate the fact that actually I couldn't support her."

"Oh Tom!" grinned Jerry. "I thought I couldn't work so well with people..."

Tom didn't immediately react. Then asked, "So why does our test automation stink?"

"Actually, it's not smelling nearly as bad now as it was," grinned Jerry. "I just wanted to see what you would say."

Tom laughed then, and the tension disappeared (as usual with Tom and Jerry). Still grinning Jerry went on, "Anyway Liz has already done a lot to improve our test automation and she is even having success in bringing testers and developers together!" He admitted, "I never managed to...Actually I didn't even try..."

Now Tom got curious, "What do you mean?" he asked. "What do you propose?"

Jerry told him how Liz had introduced him to the Test Automation Patterns and told him that the issue that was ailing test automation at Digiphonia was *TOOL-DRIVEN AUTOMATION*. Liz had also told him about the diagnostic feature, so he concluded "I plan to try it out and see what patterns it suggests. If, in order to get our automation up and running successfully, we have to invest more time or resources, I want to have your support."

"Do you already have some idea?" Tom asked.

And Jerry replied "Well, Liz is already implementing KEYWORD-DRIVEN and DATA-DRIVEN TESTING and was talking about writing a TEST AUTOMATION FRAMEWORK."

"OK," said Tom, who hadn't really understood what Jerry was talking about. "When do you want to decide on this?" he asked and added, not sounding particularly eager "Should I be there too?"

"I think it would be a good idea, and I will ask Paul too," replied Jerry. "I'll schedule a meeting with us, Liz, Jill, Tim and Ronald in a few days. I'll let you know exactly when" and he left.

Tom was still somewhat stunned and then thought "Liz here, Liz there...Could it be that Jerry has actually caught fire?" and smiled to himself.

Later that evening he called his father and told him of his suspicion.

Paul was not so surprised and answered "Well, why not? I think they would fit well together!" and added "I'll keep an eye on them"

A few days later they all met in the small conference room. On the big screen there already was the first diagnostic question. We have seen it before, but here it is again:

If you are not satisfied with your current automation, what describes the most pressing problem you have to tackle at the moment?

1. Lack of support: from management, testers, developers etc.
2. Lack of resources: staff, software, hardware, time etc.
3. Lack of direction: what to automate, which automation architecture to implement etc.
4. Lack of specific knowledge: how to test the Software Under Test (SUT), use a tool, write maintainable automation etc.
5. Management expectations for automation not met (Return On Investment (ROI), behind schedule, etc.)
6. Expectations for automated test execution not met (scripts unreliable or too slow, Tests cannot run unattended, etc.)
7. Maintenance expectations not met (undocumented data or scripts, no revision control, etc.)

They all looked at Jerry and Tom. What would they pick?

Jerry carefully read all the possible answers, and finally said "I don't think 1 to 4 are really applicable to us. Number 6 could be interesting, but as manager I would select no 5, Management expectations not met, although it could also be number 7 about maintenance." And turning to Tom he asked, "what would you say?"

Tom looked up and said, "No 5 is OK with me and if not, I understand that we can always come back, right?"

Liz nodded and flipped to the next question.

Second question (Management expectations for automation not met)
What is the main expectation that the automation hasn't met?

1. Management expected too much from automation: look up the issue UNREALISTIC EXPECTATIONS
2. The expected Return on Investment (ROI) has not been achieved. The issue that deals with this is HIGH ROI EXPECTATIONS.
3. The automation project is not on schedule (development of automation is too slow). The issue that suggests possible patterns to solve this problem is SCHEDULE SLIP.
4. The reports from automation are too difficult to analyse. Look up the issue OBSCURE MANAGEMENT REPORTS if management finds the reports unsatisfactory or INEFFICIENT FAILURE ANALYSIS if reports on failures aren't helpful.

Jerry immediately stated, "I think we can forget No. 4 in this list for now, but any one of the other answers could be correct." He asked, "what would you select, Liz?"

Liz was perplexed at being asked first, but after a short pause answered "Well, as an automator, I know that our problem is currently that maintenance is too burdensome, which would have been the other choice from the previous screen, but as I don't know what you, Tom and Jerry, expected as managers, maybe Nos 1, 2 or 3 would be more fitting..."

Tom had been listening attentively and now thought, "Ah, this is just the right moment to check if my suspicions about Jerry and Liz are correct." He took on a grave mien and interrupted her, "We want to know what you think and not what you think that we think!" he said brusquely and proceeded, "For me I believe that point 3, that we are not on schedule, is the one we should discuss in more detail."

Jerry cringed visibly at Tom's rebuttal of Liz, and replied rather defensively, "I think we should at least consider what Liz proposed. After all, if the reason we couldn't automate more test cases was due to maintenance problems, then we should first look at that issue."

Now Ronald dared to reply and said, "We should also remember that if we just continue to automate the same way as before, we won't ever have really effective and efficient automation." And added, "What good is it to have automation when we spend more time trying to find out what went wrong than we would have needed to just perform the tests manually?"

Tom had seen his suspicions confirmed and found it quite difficult to concentrate again on test automation, so it took some time for him to digest Ronald's contribution.

Only when Paul also agreed with Ronald, did he finally concede, "Hmm, maybe you're right. Let's go back and see where No 7 -Maintenance expectations not met- would bring us." But he added, "I still think that we should keep the planned schedule."

Now Liz showed the next question.

Second question: (Maintenance expectations not met)

What is the main reason for maintenance expectations not being met?

1. Maintenance costs too high. Maintenance of the automation is (considered) too expensive.
2. Not reusing existing data or other problems with test data. If it's faster (and easier) to create new data than to reuse existing data, or no-one knows what data is available to use, the issue to look at is TEST DATA LOSS. If you can only use test data once before it gets corrupted, test data is only valid for one release, or data has to be updated from release to release, then the issue to look at is INCONSISTENT DATA.

3. Other people have trouble understanding what the tests are, what the scripts do, how the automation works etc. The issue to look at is INADEQUATE DOCUMENTATION.
4. You have written all your scripts in the language and structure of your current tool, but that tool is no longer appropriate for you. The issues you should look up is TOOL DEPENDENCY.
5. Documentation is non-existent, so only the developer of the automation can work with it. The issue OBSCURE TESTS will give you a good start on how to solve this problem.
6. It's difficult to pair the automated scripts to the correct release of the Software Under Test (SUT). Look up the issue INADEQUATE REVISION CONTROL for suggestions.

Tom immediately said, "Oh it must be number 1 - the maintenance of the automated tests takes too much time and is therefore costing too much."

After looking through the other options, Jerry said, "Well it could be number 4, since the scripts are very dependent on the tool's language, but number 1 is probably closer to where we are."

So, Liz showed them the next question.

Third question (maintenance costs too high)
What costs too much when maintaining test automation?

Please select the answer that describes best where you see the highest expense.

1. Updating the automation scripts after changes have been made to the Software Under Test (SUT) and the existing tests either no longer run or give incorrect results. It takes more effort to adapt the existing scripts than to start again.
2. Refactoring the automation scripts in order to improve them or redesign them structurally.
3. Creation or maintenance of test data, where the data used in the automated tests has become "out of step" with the SUT and needs to be updated or needs to be generated from some other source.
4. People costs have not been budgeted at all or not correctly.
5. Other costs (hardware, software, networks, etc.) for maintaining automated tests have not been budgeted or are higher than expected.
6. Changing to a new tool means that we can't use any of our existing tests without major effort. See the issue TOOL DEPENDENCY.

Tim, Jill and Ronald started to whisper together, and Jerry asked them, "Well? If you already know the answer let us know too"!

All three immediately shut up and it was Ronald who answered, "Sorry, but we believe that the only possible answer would be No. 1, Updating the scripts takes too long."

Liz nodded in agreement, but Jerry mumbled loudly, “No. 4, People costs, could be the reason for No 1; I would rather look that up. What would you pick, Tom?” he asked finally.

Tom was still considering the possible options, but then said “Yes, both could be correct. Why don’t we look up both?”

Liz showed the questions coming up after no. 1:

Why do you need to update your automation scripts?

Please select the main reason to rework the scripts from the list below:

1. Minor changes to the Software Under Test (SUT) cause large changes to the automation: look up the issue *BRITTLE SCRIPTS* to get suggestions how to solve this problem.
2. Substantial changes to the SUT: you will not be able to avoid reworking your scripts. In fact, it can be the right time to switch to a better tool or a better test automation architecture (see issue *SUT REMAKE*).
3. The SUT makes automation difficult: the issue *HARD-TO-AUTOMATE* describes quite a few behavioural problems. Look it up for suggestions about which patterns could help solve your problems.
4. Changes to the automation tool(s) are causing lots of scripts to need updating: the issue *TOOL DEPENDENCY* will guide you to the patterns that can help you solve this problem.

After reading the possible answers, Jerry said, “I think here we must let the experts decide,” and, looking at Liz and the others, added, “What do you four suggest?”

Jill answered for the others. “We were here before and had decided that *BRITTLE SCRIPTS* was our issue,” she said.

“OK,” said Tom, “now please show us the page for No. 4, People costs,” he continued, looking just at Liz.

Liz complied and showed:

People costs

There is a perception that the “people” cost is too high with your automation effort.

Do you have a reasonable idea about what this cost actually is? Do you know which automation-related activities and tasks are the most costly? You don’t need exact numbers, but you do need some concrete evidence of where your costs currently are, and whether or not they are reasonable.

What costs have not been budgeted correctly?

1. Automation is done (or has been done) ad-hoc with no planning or preparation: look up the issue *AD-HOC AUTOMATION*.

2. Test execution needs manual intervention for the automation to continue. The issue MANUAL INTERVENTIONS suggests the patterns needed to resolve this problem.
3. Support for test automation is not the first priority for management, so automators lose time waiting for support from specialists (testers, database administrators, developers). Look up the issue INADEQUATE SUPPORT for help.
4. Management thinks that once automation is running there is nothing more to do (who needs maintenance?): the issue to look up in this case is UNREALISTIC EXPECTATIONS.
5. Training is needed but is difficult to get or to justify.

“Interesting,” observed Paul, who had been only listening so far. “One issue drives you to a technical problem and the other one more to a management problem! I wonder which one is the right one for us?”

“Probably both,” suggested Jerry. “The technical problem was caused by the missing support. We thought that buying the tool was all we needed to do, but we never gave a thought to developing an automation strategy and so everybody just tinkered on.”

Tom considered this for a moment and then replied, “If I understand you correctly then our answer could be AD-HOC AUTOMATION, let’s have a look!”

AD-HOC AUTOMATION (Management Issue)

Issue Summary

Automation is done ad-hoc with no planning or preparation.

“Ad hoc” means something created or done only for a particular purpose, impromptu, expedient, improvised, makeshift, “rough and ready”.

Examples

- Management thought that buying a tool was enough. Goals, architecture, resources etc. have not been planned or considered.
- Expenses for test automation resources have not been budgeted.
- There are not enough machines on which to run automation.
- The databases for automation have to be shared with development or testing.
- There are not enough tool licences.
- Automation is done “on the side” when one has time to spare.
- Support from testers, developers or other specialists has not been planned, so they have no time to help.

Questions

Who is doing automation? Is somebody in charge?

Does management know about it? Do they support it?

Resolving Patterns

Most recommended:

- **TEST AUTOMATION OWNER:** Appoint an owner for the test automation effort. If there is already a "champion", give him or her public support. The owner, once test automation has been established, controls that it stays "healthy" and keeps an eye on new tools, techniques or processes in order to improve it.
- **MANAGEMENT SUPPORT:** you need this pattern in order to be able to apply the other patterns.
- **DEDICATED RESOURCES:** helps you get the resources you need.
- **WHOLE TEAM APPROACH:** if your developer team uses an agile development process, this is the pattern of choice.

Other useful patterns:

- **SET CLEAR GOALS:** Define the automation goals from the very beginning and when you want to re-vitalise a stalled automation effort.
- **DO A PILOT:** Start a pilot project to explore how to best automate tests for the application.

TEST AUTOMATION BUSINESS CASE: this pattern may be needed when you start planning test automation or to justify making changes to existing automation that has stalled or not given sufficient benefit.

"Ouch!" said Jerry. "The first one on this list is exactly what we did!"

"You know what?" mused Paul, "I think what it's trying to say is that in fact we should have done a lot more thinking and planning before just starting our automation, but I don't see any workable suggestions on how to get out of this mess now."

"I don't know," sighed Tom. "Another suggestion was *INADEQUATE SUPPORT*. How about looking into that?"

Everybody nodded in approval, so Liz showed the page:

***INADEQUATE SUPPORT* (Management Issue)**

Issue summary

The test automation team doesn't get adequate support from management, testers or other specialists

Examples

1. Management knows too little about the benefits of test automation or about the investment needed in time or money to achieve good automation.
2. Testers don't have time to write test cases to be automated or to otherwise support the test automation team.
3. The test automation team never gets time for maintenance.
4. You need support (from IT-specialists, developers, database experts, business analysts etc), but nobody has time for you.

5. Expenses or time for training haven't been budgeted.

Resolving Patterns

Most recommended:

Automation started by one person does not often "scale up". You may need to use this experience as a learning exercise and start again with good MANAGEMENT SUPPORT. This pattern also applies if the automation work does not have a high enough priority, compared to other tasks.

When support from other people is not there, this could also be because of a failure to PLAN SUPPORT ACTIVITIES. If you end up starting again, be sure to use this pattern, which may be more difficult if your team is distributed over different geographical locations.

Other useful patterns:

- SELL THE BENEFITS: use this pattern again and again to show everyone what the automation effort can achieve with their support.
- SHARE INFORMATION: this pattern may help explain why people have no time for you. It is important to let people know why and when you need their support.
- SHORT ITERATIONS: use this pattern to work with short feedback cycles. This will let people who are supporting the automation effort know more quickly that their support is enabling you to achieve results, and this will encourage further support.

After reading this issue, Tom was still dissatisfied and noted, "I don't really see the worth of these patterns: MANAGEMENT SUPPORT, SELL THE BENEFITS, SHARE INFORMATION etc. is all just blah blah blah! There is no concrete suggestion!"

This time it was Liz who took the bait. "Yes, if you just read the pattern names, you may get that impression. On the other hand," she continued after a short pause, "there is in fact an inherent problem here: if you go too deep into the details, just like in the design patterns, you, as a manager, would be overwhelmed, but if you leave things a little fuzzy, like here, then you complain that it's not specific enough!" Finally, she concluded, "I think what these patterns mean, SELL THE BENEFITS, SHARE INFORMATION etc. is that the specialists, in this case testers and test automators, should select the patterns to implement but then they are supposed to explain the what and why to management."

Tom considered this for a while and then, assenting, he asked Liz to go back and show what the issue *BRITTLE SCRIPTS* proposed.

***BRITTLE SCRIPTS* (Design Issue)**

Issue Summary

Automation scripts have to be reworked for any small change to the Software Under Test (SUT)

Examples

Scripts are created using the capture functionality of an automation tool. If in the meantime something has been changed in the application, the tests will break unless recorded anew.

A small change to the application (such as moving something to a different screen or changing the text of a button) causes many scripts to fail because this information is embedded in the scripts for many tests.

Questions

How do you develop automation scripts? (Capture is not good long-term though ok for short complex actions.)

Is there repeated code in any scripts? (Keep your automation code DRY - Don't Repeat Yourself - put common code and actions into scripts called by other scripts.)

What kinds of changes are most likely to happen to the application? (Make your automation most flexible for those changes.)

Resolving Patterns

Most recommended:

- **TESTWARE ARCHITECTURE**: This pattern encompasses the overall approach to the automation artefacts and is best thought about right from the start of automation so that you can avoid having this issue. This pattern is implemented using:
- **ABSTRACTION LEVELS**: This is the pattern to apply if you want to delegate some of the maintenance effort to the testers. It will enable you to write test cases that are both independent from the SUT and from the technical implementation of the automation (including the tool(s)).
- **MAINTAINABLE TESTWARE**: This is the pattern to apply if you want to get rid of the issue once and for all. If you haven't implemented it yet, you may want to apply at least some aspects of this pattern.
- **MANAGEMENT SUPPORT**: This is the pattern to apply if you are missing support or resources that you need in order to develop **MAINTAINABLE TESTWARE**.
- **MODEL-BASED TESTING**: This pattern involves considerable effort at the beginning but is the most efficient in the long run. Using a test model, the test cases can be cleanly separated from the technical details. Frequently used sequences of test steps can be defined as reusable and parameterized building blocks. If the SUT changes, usually only a few building blocks need to be adapted while the test scripts are updated automatically.
- **COMPARISON DESIGN**: Design the comparison of test results to be as efficient as possible, balancing Dynamic and Post-Execution Comparison, and using a mixture of Sensitive and Robust/Specific comparisons.

Other useful patterns:

- **GOOD PROGRAMMING PRACTICES:** This pattern should already be in use! If not, you should apply it for all new automation efforts. Apply it also every time you have to change current testware.

Tom's reaction to this pattern was similar to the tester's reaction some time before. He didn't understand it, but in fact, he didn't want to have anything to do with it! He groaned, "I think you're right, Liz! This is not for me. I'd rather keep out of this technical stuff."

Jerry tried to appease him, "Yeah, I know, but I wanted you to understand what kind of problems we're facing. We can't just come over and say we need this and that simply because."

"OK, OK," replied Tom. "I understand that you think you can really improve things, so you have my approval to work on it further. But," he added after a short pause turning serious, "I expect that in a short time I will be able to see real benefits, is that clear?"

Everybody nodded and standing up he asked his father, "You coming, Dad?"

Paul didn't move, but replied, "No, I find it all very interesting, I'll stay on."

Tom was taken aback. "You won't understand all this technical stuff either."

"Sure, but then I'll ask," answered Paul with a twinkle in his eye.

Tom knew better than to argue and left the room slightly annoyed.

Jerry said, "I need a cup of coffee, what about you?" he asked as he stood up. Then he added, "and then we get down to business!"

1.1.11. Finding the Best Solution

Searching the issues,
how to find the right patterns?
The diagnostic!

On returning, bringing coffees and teas for everyone, Jerry complained, “These patterns are not as easy to use as I was hoping,” and continued, “You get so many suggestions that are so interlaced that you don’t know where to start!”

“Yes,” agreed Liz smiling, “that’s actually a good description of test automation!” She added, “We should get down to business just the same. Where do we go from here?”

“You’re right of course, Liz,” said Jerry, his ears slowly colouring. “Where did we stop? Can you show us the recommended patterns from that last issue? I think it was *BRITTLE SCRIPTS*. ”

Liz brought it up on the screen again (we show now only the relevant resolving patterns)

***BRITTLE SCRIPTS* (Design Issue)**

Resolving Patterns

Most recommended:

- **TESTWARE ARCHITECTURE**: This pattern encompasses the overall approach to the automation artefacts and is best thought about right from the start of automation so that you can avoid having *this issue*. This pattern is implemented using:
- **ABSTRACTION LEVELS**: This is the pattern to apply if you want to delegate some of the maintenance effort to the testers. It will enable you to write test cases that are both independent from the SUT and from the technical implementation of the automation (including the tool(s)).
- **MAINTAINABLE TESTWARE**: This is the pattern to apply if you want to get rid of the issue once and for all. If you haven't implemented it yet, you may want to apply at least some aspects of this pattern.
- **MANAGEMENT SUPPORT**: This is the pattern to apply if you are missing support or resources that you need in order to develop **MAINTAINABLE TESTWARE**.
- **MODEL-BASED TESTING**: This pattern involves considerable effort at the beginning but is the most efficient in the long run. Using a test model, the test cases can be cleanly separated from the technical details. Frequently used sequences of test steps can be defined as reusable and parameterized building blocks. If the SUT changes, usually only a few building blocks need to be adapted while the test scripts are updated automatically.
- **COMPARISON DESIGN**: Design the comparison of test results to be as efficient as possible, balancing Dynamic and Post-Execution

Comparison, and using a mixture of Sensitive and Robust/Specific comparisons.

Other useful patterns:

GOOD PROGRAMMING PRACTICES. This pattern should already be in use! If not, you should apply it for all new automation efforts. Apply it also every time you have to change current testware.

“Fine,” said Jerry “now let’s compare it with the issue that you told me describes exactly our problem.”

“You mean *TOOL-DRIVEN AUTOMATION*, right?” replied Liz, and she opened also this issue (here again we show now only the resolving patterns)

***TOOL-DRIVEN AUTOMATION* (Process Issue)**

Resolving Patterns

Most recommended:

- **TOOL INDEPENDENCE**: Separate the technical implementation that is specific for the tool from the functional implementation.
- **GOOD PROGRAMMING PRACTICES**: Use the same good programming practices for test code as in software development for production code.
- **LAZY AUTOMATOR**: Lazy people are the best automation engineers.
- **TEST AUTOMATION OWNER**: Appoint an owner for the test automation effort.
- **TESTWARE ARCHITECTURE**: Design the structure of your testware so that your automators and testers can work as efficiently as possible.
- **WHOLE TEAM APPROACH**: Testers, coders and other roles work together on one team to develop test automation along with production code.

Other useful patterns:

- **LOOK FOR TROUBLE**: Keep an eye on possible problems in order to solve them before they become unmanageable.
- **MAINTAINABLE TESTWARE**: Design your testware so that it does not have to be updated for every little change in the Software Under Test (SUT).
- **SET STANDARDS**: Set and follow standards for the automation artefacts.

GET TRAINING: Plan to get training for all those involved in the test automation project.

Jerry observed “Look, there are two patterns recommended by both issues: GOOD PROGRAMMING PRACTICES and MAINTAINABLE TESTWARE, which has also

been recommended from another issue. I guess that could be our lead. Liz, can you please show it to us again?"

MAINTAINABLE TESTWARE (Design Pattern)

Implementation

Some suggestions:

- There are many options to make and keep the testware maintainable, but to adopt a GOOD DEVELOPMENT PROCESS and GOOD PROGRAMMING PRACTICES is a very good bet: what works for software developers works for test automation just as well!
- For example, if your scripts are mainly "stand-alone" without much re-use, and automators are frequently re-inventing similar or even duplicated scripts or automated functions, then GOOD PROGRAMMING PRACTICES are needed, particularly DESIGN FOR REUSE and OBJECT MAP.
- Implement DOMAIN-DRIVEN TESTING: test automation works best as cooperation between testers and automation engineers. The testers know the SUT, but are not necessarily adept in the test automation tools. The automation engineers know their tools and scripts, but may not know how best to test the SUT. If the testers can develop a domain-specific language for themselves to use to write the automated test cases, the automation engineers can implement the tool support for it. In this way they will each be doing exactly what they do best. The advantage for the automation engineers is that in this way the testers will maintain the tests themselves leaving the engineers more time to refine the automation regime, and to solve technical maintenance problems.
- For example, if you have structured and reusable scripts, but there is a shortage of test automators to produce the automated tests (or they are short of time), this pattern gives the test-writing back to the testers, once the automators have constructed the infrastructure for the domain-based test construction. Other useful patterns are ABSTRACTION LEVELS and KEYWORD-DRIVEN TESTING.
- Don't forget to actually MAINTAIN THE TESTWARE: maintainability alone is not enough, it must actually *be* maintained.

Potential problems

Don't wait too long to build maintainable testware - this is best thought of right at the beginning of an automation effort (although it is also never too late to begin with improvements).

After reading the pattern once more, Jerry said "As a developer, I think we should also have a look at GOOD DEVELOPMENT PROCESS and GOOD PROGRAMMING PRACTICES, and I agree with the assertion that what works for software development should work for test automation as well."

They all nodded so the next pattern on the screen was GOOD DEVELOPMENT PROCESS.

GOOD DEVELOPMENT PROCESS (Process Pattern)

Description

Test automation is development, so you should adopt a process similar to the software development process (assuming it is a good development process). Development of test code works very well if it is an agile process.

Implementation

Learn about software development principles, if you are going to be working directly with the scripting tools. There may be courses on programming from your local university, for example. Test code needs to be well structured, with small self-contained modules that call other reusable modules. Changes to the software should impact the smallest number of test modules. Scripts and all other testware should be under version control (Configuration management).

Agile development in general uses the following elements: (which work very well when developing automation)

- **SHORT ITERATIONS:** you don't plan everything up front, but break up the work-load into short iterations. At the beginning of the iteration you plan to do only what you can completely finish. At the end of the iteration you have working software that you could deploy to production. The customer can check if you delivered what he or she had in mind. If not, you adapt it as required in the following iterations. In the case of test automation, your code will be scripts and your customers will be the testers, so it works the same way.
- Everyone on the team is responsible for the quality of the developed software. For test automation this means that team members:
 - PAIR UP
 - TEST THE TESTS
 - MAINTAIN THE TESTWARE
 - SET STANDARDS

You can document your test automation process in a Wiki.

Jerry read through the pattern, but then, instead of immediately deciding what to do, he turned to Ronald and said, "I see five suggested patterns. I don't think we'll be able to do them all at the beginning. Which ones are the most important for you, Ron?"

Ronald didn't hesitate. "First SET STANDARDS and then MAINTAIN THE TESTWARE," he said.

"Are you with Ron?" asked Jerry turning to the others and they all nodded.

"OK," he approved, "I'm also with you on that. We'll meet in the next few days to start defining our test automation standards. I'll send you the link to our development standards. I think we can definitely build on them."

"What about MAINTAIN THE TESTWARE?" asked Jill and went on, "we already do a lot of maintenance, but it's currently out of plan so that we are always doing it with a sore conscience because we think we should be testing instead."

"Hmm," replied Jerry and turning to Paul he asked, "What do you think, Paul?"

"I'm with you," said Paul. "I'll talk to Tom. He usually lets me decide what's needed for testing. I'm beginning to think that pretty soon Liz will not be the only test automator that we will have hired."

Liz's first thought at that was "Wow, I could ask Jack if he would be interested! It would be awesome to work together again!"

"OK, so that's resolved," agreed Jerry. "Now let's see what kind of programming practices are recommended. Liz, please?"

GOOD PROGRAMMING PRACTICES (Design Pattern)

Description

Scripting is a kind of programming, so you should use the same good practices as in software development.

Implementation

If you don't have an automation engineer, have developers coach the automation testers. Important good practices are:

- DESIGN FOR REUSE: Design reusable testware.
- KEEP IT SIMPLE: Use the simplest solution you can imagine.
- SET STANDARDS: Set and follow standards for the automation artefacts.
- SKIP VOID INPUTS: Arrange for an easy way to avoid void inputs.
- INDEPENDENT TEST CASES: Make each automated test case self-contained.
- ABSTRACTION LEVELS: Build testware that has one or more abstraction layers.
- Separate the scripts from the data: use at least DATA-DRIVEN TESTING or, better, KEYWORD-DRIVEN TESTING.
- Apply the "DRY" Principle (Don't Repeat Yourself). Also known as "DIE" (Duplication is Evil).

Put your best practices in a Wiki so that both testers and developers profit from them.

After reading the pattern, Jerry nodded, "We are definitely on the right track! See here again it suggests implementing SET STANDARDS!"

At this point Paul interrupted him, asking "What are ABSTRACTION LEVELS?"

Before Liz could answer, Jill butted in with the hamburger example and at the end everybody, Liz included, was chuckling.

Paul asked to see the pattern too and so Liz flipped over to the appropriate page.

ABSTRACTION LEVELS (Design Pattern)

Description

The most effective way to build automated testware is to construct it so that it has one or more abstraction layers (or levels). For example, in software development, the code to drive the GUI is usually separated from the code that actually implements the business functionality and also separated from the code that implements access to the database. Each part communicates with the others only through an interface. In this way each can be individually changed without breaking the whole as long as the interface is not changed (this is the theory, it is not always practiced).

For test automation this means that the testware is built so that you write in the scripting language of the tool only technical implementations (for instance scripts that drive a window or some GUI-component of the SUT). This is the lowest layer. The test cases call these scripts and add the necessary data. This is the next layer. And if you implement a kind of meta-language you get another layer. As for software code, the charm of abstraction layers is that since the layers are independent of each other they can be substituted without having to touch the other layers. The only thing that has to be maintained is the interface between them (how the test cases are supposed to call the scripts).

By separating the technical implementations in the tool's scripting language from the functional tests, you can later change automation tools with relative ease, because you will only need to rewrite the tool-specific scripts. Also if you keep the development technicalities apart from the test cases even testers with no development knowledge will be able to write and maintain the testware. And they can start writing the tests even before the SUT has been completely developed. Another advantage is that you can reuse the technical scripts for other test automation efforts.

Implementation

There are different ways to implement abstraction layers. Which to choose depends on how evolved your test automation framework is.

- DATA-DRIVEN TESTING means that you separate the data from the execution scripts (drivers). In this case you must take care to correctly pair the data to the drivers.
- In KEYWORD-DRIVEN TESTING you specify a keyword that controls how the data is to be processed. Generally, keywords correspond to words in a domain-specific language (for instance insurance or manufacturing). Normally used for DOMAIN-DRIVEN TESTING.
- In MODEL-BASED TESTING you create a test model of the SUT. Typically, the modelling of test sequences starts at a very abstract level and is later refined step by step. From the model, a generation tool can automatically create test cases, test data, and even executable test scripts.
- Use TOOL INDEPENDENCE to separate the technical implementation that is specific for the tool from the functional implementation

Potential problems

Building a good TESTWARE ARCHITECTURE takes time and effort, and should be planned from the beginning of an automation effort. There is a temptation (sometimes encouraged by management) to "just do it". This is fine as a way of experimenting and getting started, for example if you DO A PILOT. However, you soon find that you have many tests that are not well structured and get STALLED AUTOMATION with high maintenance costs, but now you are "locked in" to the wrong solution if you are not aware of the importance of abstraction levels.

"Yes," agreed Paul after reading the pattern. "Jill is right that your explanation with the hamburgers hits the nail on the head. Once you understand what really is meant, it becomes much clearer."

"OK," continued Jerry, "here are some more recommendations for KEYWORD-DRIVEN TESTING and DATA-DRIVEN TESTING. You have already started on both, so I think we should keep on with it. Should we consider standards for them too?"

"Yes, of course," answered Tim and all the others assented.

Then Ronald asked, "What about the other suggestions? DESIGN FOR REUSE and KEEP IT SIMPLE look like good things to me," he said, and added, "maybe we should put them in the standards too, what do you think?"

Jerry liked the idea and said "Yes, that's good, I'm for it." Jill, Tim and Liz nodded in approval too.

"OK," concluded Jerry, "but now I have another question and this time I want you, Liz, to answer. With AutomatePro you can, as far as I know, implement, run and check automated tests. Can you explain then what we would need a TEST AUTOMATION FRAMEWORK for?"

Finally, Liz was in her element. She beamed. "Yes, that's exactly the point: think about the hamburger, I mean ABSTRACTION LEVELS. At the moment, we have everything in the tool. What happens if we have to change to a different tool? Imagine that you could get hamburgers only from one specific vendor and no one else, imagine that you couldn't even make them at home."

Just thinking about it, Ronald turned paper white.

Liz went on, "With a framework we can reduce our dependence on the tool because we can manage most of the testing knowhow outside of the tool. In the tool we would only leave the strictly technical scripts or keywords." After a short pause she concluded smiling, "In this way changing the tool would still be a lot of work, but we wouldn't be starting from scratch. Just as with hamburgers we could just get our hamburger at some other place."

"Ah," interrupted Paul, looking serious, "maybe that would be just the right way to get rid of all our automation problems: we change the tool and restart from scratch, but this time following the patterns!"

All the others just stared at him in consternation and he immediately followed up with a big grin, "Just joking!" he said. Then he asked, "Liz, can you show us the pattern?"

"My pleasure," said Liz (and really meant it).

TEST AUTOMATION FRAMEWORK (Design Pattern)

Description

Using or building a test automation framework helps solve a number of technical problems in test automation. A framework is an implementation of at least part of a testware architecture.

Implementation

Test automation frameworks are included in many of the newer vendor tools. If your tools don't provide a support framework, you may have to implement one yourself.

Actually it is often better to design your own TESTWARE ARCHITECTURE, rather than adopt the tool's way of organising things - this will tie you to that particular tool, and you may want your automated tests to be run one day using a different tool or on a different device or platform. If you design your own framework, you can keep the tool-specific things to a minimum, so when (not if) you need to change tools, or when the tool itself changes, you minimise the amount of work you need to do to get your tests up and running again.

The whole team, developers, testers, and automators, should come up with the requirements for the testing framework, and choose by consensus. If you are comparing two frameworks (or tools) use SIDE-BY-SIDE to find the best fit for your situation.

A test automation framework should offer at least some of the following features:

- Support ABSTRACTION LEVELS.
- Support use of DEFAULT DATA.
- Support writing tests.
- Compile usage information.
- Manage running the tests, including when tests don't complete normally.
- Report test results.

You will have to have MANAGEMENT SUPPORT to get the resources you will need, especially developer time if you have to implement the framework in-house.

Potential problems

It is not necessarily easy to acquire or make a good test automation framework, and it does take effort and time. But it is very worthwhile when done well.

If you intend to develop a framework in-house make sure to plan for the necessary resources (developers, tools etc.) otherwise you could end up with a good framework that must be abandoned because for instance the

only developer leaves the company. See issue [KNOW-HOW LEAKAGE](#) for more details.

Paul read the description a couple of times, but then had to ask “I still don’t really understand what a framework is. Can somebody explain it to me?”

Liz, nodding at Jerry as if to ask permission, started, “Actually a framework can be a lot of things: programs, code libraries, data, standards - whatever is needed to make doing something easier or faster.” She stopped briefly and then went on, “Think for instance of a kitchen, that’s a good example for a framework.”

Jill stared at her. Tim smirked and said loud to himself “Another of Liz’s freak comparisons...”

Liz continued unruffled, “Yes, a kitchen! In a kitchen you have a lot of stuff that helps you do your job more easily or faster. Think of all the different kitchen machines from a simple mixer to a dishwasher to a microwave! Or think of all the different pots and pans or knives or china.” Then she added, “Think of how complicated it suddenly gets when you are camping, and you don’t have your kitchen framework along, or if you try to cook in someone else’s kitchen.” Finally, she concluded, “In our case, with test automation, a framework helps do standardized jobs like preparing initial environments for the test cases or extracting data from a database and other things. With a good framework, we can focus on automating test cases instead of having to do such chores again and again.”

At this point Ronald, who had listened attentively to Liz’s explanations, countered, “I’m not so convinced that we need our own framework.” Seeing that he had everyone’s attention, he continued, “AutomatePro supports us with implementing, testing, running and reporting results of the test cases. The tool serves us well and I see no reason to change it. We should use the resources that we already have to automate as much as possible instead of developing a framework that will just do what we, with the tool, already have!”

Liz was visibly taken aback, but managed to reply, “Yes, but then we will be bound hand and foot to this vendor.”

“Yes, but so what!” said Ronald. “We pay quite a large sum for their licences. Why should we also implement and don’t forget, maintain, something which we have already paid for?” He went on even more forcefully, “Isn’t this also a case of ABSTRACTION LEVELS? Our job is to develop Digiphonia-apps and the vendor’s job is to develop tools and frameworks for test automation! And there is another pattern that we shouldn’t forget: DON’T REINVENT THE WHEEL.”

Jerry replied, “You have a point there! We should discuss in more detail the advantages or disadvantages of developing our own framework. And I think we should also hear from the developers on this. Currently they are using a kind of rudimentary framework to run the unit tests and, for the same reason, they have already implemented quite a few APIs, so maybe we could just build from there. In any case they can help decide if it makes sense to implement something ourselves.”

Liz was shocked and disappointed that she hadn't been able to convince them on the spot but realized that this was in fact a viable solution, so she put on a good face and replied, "Yes, that's a good idea! When do we start?"

Jerry looked affectionately at her and replied jokingly, "Would tomorrow be soon enough?"

Liz smiled back and added, "I think another advantage of involving the developers would be that then we would also continue to implement the pattern WHOLE TEAM APPROACH."

Jerry replied, "What's that about? I remember you mentioning that one before."

Liz immediately showed him the appropriate page:

WHOLE TEAM APPROACH (Process Pattern)

Testers, coders and other roles work together on one team to develop test automation along with production code.

Context

This pattern is most appropriate in agile development but is effective in many other contexts as well. This pattern is not appropriate if your team consists of just you.

Description

Everyone collaborates to do test automation. Developers build unit test automation along with production code. Testers know what tests to specify, automators or coders help to write maintainable automated tests. Other roles on the team also contribute, eg, DBAs, system administrators.

Implementation

If you are doing agile development, you should already have a whole-team approach in place for software development, testing and test automation.

If you are not doing agile, it is still very helpful to get a team together from a number of disciplines to work on the automation. In this way you will get the benefit of a wider pool of knowledge (SHARE INFORMATION) which will make the automation better, and you will also have people from different areas of the organisation who understand the automation.

Potential problems

If people are not working on the automation as a FULL TIME JOB, there may be problems as other priorities may take their time away from the automation effort.

Another possible problem occurs when many DevOps teams develop test automation independently (look up the issue LOCALISED REGIMES)

Liz said, "We think that implementing this would avoid problems like automating test cases already covered by unit tests and enable us to find out sooner what the developers are planning to change."

Jill butted in, "Now we find out about changes only when the tests suddenly start to fail."

"Well I believe that shouldn't be a problem," affirmed Jerry and added, "Every morning we have a stand-up meeting, why don't one of you come too? Maybe every day someone else so that in the long run both teams get to know each other."

Ronald agreed and proposed, "Then we should also have a developer coming to our meetings, so that they know first-hand the problems that we face."

"Good idea!" agreed Jerry, "We'll do that!"

It had turned late, and Paul said "I think that's enough for now. Let's go for lunch."

Jerry stood up and said, "As soon as I have all the pertinent people together, I'll send you all an invitation to a standards meeting. Thanks for your participation. I believe we are heading in the right direction now."

After general nods of approval, they started to pack their stuff and leave the room.

Just as Liz was getting up, Paul asked her "Are you still interested in a D-eatles concert? We're playing this week-end."

"Oh, yes!" she replied, delighted, "I would love to come. Shall I tell my jazz band about it too?"

"Of course," answered Paul and added, "and bring your instruments: maybe we'll have a small jam session all together!"

"Awesome!" exclaimed Liz, "That would be great!"

Jerry had stopped to listen and was becoming more and more startled and finally said to Liz, "I didn't know that you play in a jazz band. What do you do? You have a beautiful voice, are you the singer?"

Liz laughed, "No, we have a much better lead singer. I play the bass and sing only in the refrains." She then asked shyly, "Are you coming too?"

Jerry replied, his ears turning crimson, "Yes of course, I never miss a concert!"

Paul was watching them and, grinning inwardly, well knowing that Jerry had never been a fan of Beatles music, thought "Tom is right. Jerry has really been bitten!"

1.1.12. Not Only Standards

SET STANDARDS is done.
WHOLE TEAM APPROACH is working.
Getting better, right?

The next day Jill went to the developers' stand-up meeting. Having 'made friends' at lunch, it felt quite natural. Everybody was supposed to tell briefly what they had just done and what they planned to do that day. When it was Jill's turn and she told them what she had planned to test, Jim asked her, if possible, to first test a bug fix he had just checked in.

"It was a difficult fix," he said, "and now I still know exactly what I have done. I would really appreciate if you could check it right away."

"Sure," answered Jill and thought to herself, "this pattern, WHOLE TEAM APPROACH seems to work and not just for test automation!"

When Jill announced the next day that she had not found any bugs in Jim's bug fix, he beamed and gave her a cute little candy box as a thank you. She was really embarrassed in accepting it and said, "This is really a kind thought, Jim, but I only did my job." And looking also at the other developers, she added, "Please don't follow his example. Does anybody else need some quick testing?"

The answer came from an unexpected source: Leroy, who had always been quite dismissive with her, asked with a sheepish smile, "Yep, yesterday I also made some complex changes. It would be nice if you could test them as soon as possible."

Jill couldn't mask her surprise and, with a wide smile, replied, "Of course! I'll stay after the meeting and you can tell me exactly what I should check."

When the other testers discovered how friendly the developers had been, they also started to appear at the stand-up meetings.

As for Jill, she was quite happy to find out that the developers, and especially Leroy, were not as 'scary' as she had imagined. On the contrary, more than once she had been able to help them design not only the unit tests, but even the actual code. And in exchange the developers were starting to inform her and the other testers when they planned to 'disrupt' the automated tests.

Sometime later Jill happened to relate these experiences to Liz, who immediately noted, "Yes, some of the test automation patterns are good not only for automation! I see that you all are applying SHARE INFORMATION very successfully - did you look that one up?"

Jill hadn't, but decided to do so as soon as possible.

Another advantage of the closer collaboration between the teams was that now Liz and the testers worked on the same iterations as the developers, so they were not

only applying the pattern WHOLE TEAM APPROACH, but also SHORT ITERATIONS and ASK FOR HELP

But let's go back in time. The standards meeting had been scheduled for the afternoon of the Thursday, after the big diagnostic meeting. In addition to the testers, the developers Leroy, Jim and Tabisha had also been invited to the small conference room.

When they came in, the pattern SET STANDARDS was already being displayed on the big screen, ready for scrolling down:

SET STANDARDS (Process Pattern)

Set and follow standards for the automation artefacts.

Context

This pattern is appropriate for long-lasting automation. It is essential for larger organisations and for large-scale automation efforts. This pattern is not needed for one-off scripts.

Description

Set and follow standards: otherwise when many people work on the same project it can easily happen that everyone uses their own methods and processes. Team members do not "speak the same language" and hence cannot efficiently share their work; you get OBSCURE TESTS or SCRIPT CREEP.

As an extra bonus, standards make it easier for new team members to integrate into the team

Implementation

Some suggestions for what you should set standards for:

Naming conventions:

- Suites: the names should convey what kind of test cases are contained in each test suite.
- Scripts: if the names are not consistent, an existing suitable script may not be found so a duplicate may be written.
- Keywords: it's important that the name immediately conveys the functionality implemented by the keyword.
- Data files: it should be possible to recognize from the name what the file is for and its status.
- If you implement OBJECT MAP, the right names facilitate understanding of the scripts and enable you to change tools without having to rewrite your entire automation scripts.
- Test data: if possible use the same names or IDs in all data files. It will facilitate reuse.

Organisation of testware:

- Test Definition: Define a standard format or template to document all automated tests, for example as a standard block of comment, where

the information is presented in a consistent way across all tests (e.g. same levels of indentation). This should contain the following:

- Test case ID or name.
- What this test does.
- Materials used (scripts, data files etc.).
- Set-up instructions.
- How it is called (input variables if any).
- Execution instructions.
- What it returns (including output variables).
- Tear-down instructions.
- Length (how long it takes to run).
- Related tests.
- TEST SELECTOR tags.
- Any other useful information such as EMTE (Equivalent Manual Test Effort - how long this test would have taken manually).
- File and folder organization and naming conventions.

Other standards:

- Documentation conventions for scripts and batch files.
- Coding conventions for scripts.
- Data anonymization rules.
- Develop a TEMPLATE TEST.

Other advice:

- Standards should be reviewed periodically in order to adjust or enhance them.
- Put your standards in a Wiki so that everybody can access them at any time.
- If something has to be changeable, use a translation table so that the scripts can stay stable.
- Allow exceptions when needed.
- Setting standards for test data, for example using the test ID as the customer's name, can help to VISUALIZE EXECUTION as a way of monitoring test progress.

Potential problems

When devising standards, it is useful to get input from a selection of people who will be using the automation, to make sure that the conventions adopted will serve all of its users well. An additional benefit of getting others involved is that they will be more supportive of something that they helped to devise. Not many people like having standards imposed arbitrarily when they can't see the reason for them.

Once you have settled on your standards, however, then you need to make sure that everyone does use them in a consistent way, and this will take effort.

When all participants had read through it all, Paul said, "Before we start defining standards, we should decide how or who is going to make sure that they are being applied." He continued with a grimace, "I'm not going to be the one!"

Jerry laughed, "No, I didn't expect you to, but you are right. Any proposals?"

Tabisha was the first to answer "I think it should be one of the testers or automators. None of us developers have much idea of what they're doing, so how could we know if they are using the standards correctly?"

Now Ronald spoke up, "Ok, I'll do it, but I suggest, as the pattern advises, that we have a standards meeting regularly, let's say every month or two, to check if the standards have worked or if they should be updated."

"Good idea," said Jerry, "we'll do that" and he added, turning to Liz and the other testers, "Do you agree at having Ronald not only as Test Manager, but also as a 'watch dog'?"

Everybody assented, so he continued, "Now let's get to work. Did you examine the development standards I sent you some days ago?"

Liz had immediately looked them up. They contained really only prescriptions for how to write clean code and so she and the testers had already discussed how much more they would need for their own standards.

What followed was a heated debate on what kind of standards to set. There were two schools of thought: one wanted to prescribe everything as minutely as possible and the other thought that people should be free to 'stretch' the standards when they found it useful. Ronald belonged to the 'prescribers' and Jill and Tim to the 'stretchers'. Liz was somewhat in between.

The developers didn't say anything but were often giggling between themselves: they had experienced similar discussions when they had defined their own standards.

Finally, Paul had had enough and butted in, "I think the problem is that we are talking about something undefined and so each one of us interprets it differently. I propose that each one of you, and not only testers and test automators, writes down what he or she thinks are important standards. Then we can discuss more pragmatically about what makes sense and what doesn't"

Jerry distributed small cards and added, "Please write one proposal per card and then pin them on the board."

As if Paul had said a magic word, everyone went quiet and started writing. Even the developers, because they wanted the bug reports from the automated tests to also be standardized.

When everyone was finished, Jerry asked them to categorize the cards and, that done, he picked up the first one, read it aloud and asked them what they thought about it. Now the discussion was much more objective and so they were able to work through a good third of the proposals, before it was time to stop.

Jerry asked turning to Jill, "Jill, can you write up what we have already defined?"

Jill nodded, and he added, "We'll continue next week Monday so that you all can sleep over the standards that we have already established."

When the others were gone, Jerry turned to Paul and said "We should have used the cards from the very beginning. We would have saved a lot of time!"

"Yes," answered Paul, "but then they would have thought that we were playing the usual 'management consultant games'. Now they've understood why we've done it that way," he explained, "That's an old trick of mine: first let them fight and then, after they see how futile that is, they are eager to collaborate, and things get done much more quickly!"

On Saturday evening Liz and her band were entering the small theatre by the stage door. They had been asked by Paul to leave their instruments back stage and then come and sit in the front. He had reserved seats for them in the very first row and, apparently by chance, Jerry was sitting right next to Liz. On his other side was Tom with his wife, Veronica.

They had agreed to play a couple of songs before the break and, when it was finally their turn, Paul called them all up on the stage and announced to the surprised public that there would be a short intermezzo with the Happy Tiger Jazz Band and presented each with their instrument.

They played two pieces and were rewarded with long and loud applause. After returning their instruments and joining the others for the break, Liz asked Jerry, "So? How did you like it?"

Jerry beamed and said, "Great! I've never really liked jazz and I usually hate the saxophone, but Tigg plays it with so much soul that it becomes mellow and warm instead of shrill as I expected! And she really does have an exceptional voice" He then continued, his ears turning crimson "and I loved your solo with the bass. You're really good - have you ever considered playing Bach?"

Liz was impressed that Jerry could freely admit to not liking jazz. She had feared that he would pretend just to please her. She smiled and answered "I'm afraid I'm not patient enough for Bach. With jazz I am free to improvise as I feel in that moment. I don't think I'd like to do the same thing every time." And after a short reflection she added, "You know, that's probably also the reason I feel like a LAZY AUTOMATOR: I just don't like to do the same stuff again and again!" Finally, she concluded jokingly, "I think, to get me to play Bach, you would have to jazz it up!"

At these words Jerry's eyes first lit up and then glazed over, and he sank into deep thought. Tom was nearby and before Liz could say anything he signed her to be quiet. "Leave him alone, that's his 'new-app-look,'" he explained. "Don't you see all the little wheels in his brain running at speed? He's probably calculating how to adapt his algorithm to some new idea. By the way what were you talking about?"

Liz replied, "I joked that he would have to jazz Bach up for me to play it."

Tom nodded. "I see. I think we are going to get a whole new family of apps!" he said, grinning widely.

At the end of the concert Paul invited the Happy Tiger Jazz Band back on the stage and all together they played the Beatles' Yesterday to a standing ovation.

Later, together at a restaurant, Tom was talking to Tigg, "You have a fantastic voice," he was saying. "Did you ever consider singing in a choir?"

"Oh, but I do," replied Tigg. "I regularly sing in my church's gospel choir. Why do you ask?"

"Well, Rony and I sing in a choir too and we would really love to have a voice like yours in our choir."

Tigg smiled, "Well, it depends on what kind of music you do."

Tom answered, "We sing a lot of madrigals and mostly old music from Italy, France or England."

Tigg didn't seem very eager, so Tom went on "Why don't you come to our next rehearsal Tuesday evening? Then you can decide if you like it or not."

Veronica, who had been listening on the side, butted in, "Oh, please do come...please..."

Tigg reflected a short moment and then nodded, "Ok, but only as a test, and only this one time."

Back at work, a week later they were finally finished with defining their standards. As suggested by the pattern, they had established all kinds of naming conventions. They had also convinced the developers to give unique names to every single GUI-object and to select names that were also significant for the testers. In this way the objects in the object repository could just keep the development names: developers and testers would 'speak' the same language!

For the code, they more or less adopted the same rules that the developers had been using themselves, but they also designed a template function to use when implementing new keywords so that they could immediately see what they had to document and how.

They also heeded the proposals from the developers about the bug reports and decided together how to redesign them.

What they had left open was how to name test data files. Liz had proposed to give them the same name as the test cases that were using them. This would have had the advantage of immediately knowing what belonged to which test case. Ronald had countered that some data files were used by a lot of test cases and that it would have been crazy to create so many copies. They agreed to decide from case to case. In the next meeting they could then examine how the two systems had worked out.

1.1.13. Frameworking

Just like a kitchen
helps you do a better job,
so does a framework!

The next pattern to consider was TEST AUTOMATION FRAMEWORK, and whether or not to build an automation framework for their tests. Liz, Ronald, Tim, Jill, Jerry, Leroy and Jim came together for the first meeting the week after the standards had been defined.

Jerry opened it by showing the pattern:

TEST AUTOMATION FRAMEWORK (Design Pattern)

Description

Using or building a test automation framework helps solve a number of technical problems in test automation. A framework is an implementation of at least part of a testware architecture.

Implementation

Test automation frameworks are included in many of the newer vendor tools. If your tools don't provide a support framework, you may have to implement one yourself.

Actually, it is often better to design your own TESTWARE ARCHITECTURE, rather than adopt the tool's way of organising things - this will tie you to that particular tool, and you may want your automated tests to be run one day using a different tool or on a different device or platform. If you design your own framework, you can keep the tool-specific things to a minimum, so when (not if) you need to change tools, or when the tool itself changes, you minimise the amount of work you need to do to get your tests up and running again.

The whole team, developers, testers, and automators, should come up with the requirements for the testing framework, and choose by consensus. If you are comparing two frameworks (or tools) use SIDE-BY-SIDE to find the best fit for your situation.

A test automation framework should offer at least some of the following features:

- Support ABSTRACTION LEVELS.
- Support use of DEFAULT DATA.
- Support writing tests.
- Compile usage information.
- Manage running the tests, including when tests don't complete normally.
- Report test results.

You will have to have MANAGEMENT SUPPORT to get the resources you will need, especially developer time if you have to implement the framework in-house.

Potential problems

It is not necessarily easy to acquire or make a good test automation framework, and it does take effort and time. But it is very worthwhile when done well.

If you intend to develop a framework in-house make sure to plan for the necessary resources (developers, tools etc.) otherwise you could end up with a good framework that must be abandoned because for instance the only developer leaves the company. See issue [KNOW-HOW LEAKAGE](#) for more details.

While the others were still reading the pattern, Leroy butted in angrily, “I find it highly irritating that some words are, with no apparent reason, written in capital letters. What should that mean? Why do you consider this site so important when it can’t even write correctly? What...”

Jerry interrupted him before he could go on with his rant. “Calm down, Leroy! We asked you here because your opinions are usually well thought out. I see that we took too much for granted.” And turning to Liz he added “Liz, you introduced me to the patterns. Can you please explain them also to Leroy and Jim?”

Liz nodded and after a short reflection said, “You must already know and apply the design patterns from the Gang of Four⁶, right?”

After both Leroy and Jim assented, she went on, “The patterns we are talking about here are for system test automation and are somewhat fuzzier than the patterns you are used to. The words in capital letters that you find so irritating are actually pattern names and links to those patterns. The main difference to code patterns is that these patterns are not prescriptive. In system test automation you have a much wider range of issues and they can’t be solved just by a code snippet. Usually, depending on the context, these patterns suggest different solutions and, as you can see here, that can mean other patterns.”

Leroy seemed unconvinced and interrupted, “I don’t understand then why you call them patterns if they are just suggestions.”

Before Liz was able to answer, Tim butted in. “I can tell you. I was at a testing conference and there I learned about them. They are called patterns because a lot of practitioners when confronted with similar issues came up with the same solutions. Until the patterns were published everyone had to invent the same ‘wheels’ again and again. With the patterns, you can build on the experience of others.”

This made sense to Leroy and after a short time he said, “Ok, I think I get it. You could call it a kind of primitive expert system, right?”

Now it was Tim who looked as if Leroy had just spoken Greek!

⁶ See References in Appendix

Jerry laughed and said, turning to Tim, “An expert system is a computer system that emulates the decision-making ability of a human expert⁷. There is usually some kind of database with the expert knowledge and rules which the system can use to find the solutions. In our case you have test automation issues and each issue presents one or more patterns as expert solutions. So yes, you can imagine it as a primitive expert system.”

Tim hadn’t really understood, and he didn’t dare answer, but he promised himself to look up expert systems as soon as the meeting was finished.

Leroy had one more question, “And why is *KNOW-HOW LEAKAGE* in italics?”

This time it was Jill who answered. “That’s an issue.” she said with a big smile, happy to be able to show that she also knew all about patterns “The italics are so you can distinguish it from a pattern. The issues are problems or things that you need to do, and to help with or solve an issue, the wiki points you to patterns that might help.”

After asking Leroy and Jim if they had more questions (they didn’t), Jerry continued, “You have now all read the framework pattern. We want to discuss if we should implement it for our test automation.” And he added, “We have two contrary opinions: from Liz and from Ronald. Can you two please explain again why you are for or against it?”

Ronald immediately set off and more or less repeated what he had said before: why should they implement something that actually was already conveniently embedded in the tool they were using? He thought that they should concentrate on automating test cases and not on writing frameworks.

As for Liz, she had taken the time the week before to study in detail what kind of support the tool truly offered. She wanted to be able to show that by developing an in-house framework they would work so much more efficiently that it would pay for itself very quickly. Actually, she thought that simply being independent from the tool would already be enough to justify the framework, but she realized that that alone would not help her convince the others, and especially not Ronald.

She started with a question: “Who is going to automate the test cases?”

The others looked at her confused. Ronald replied, rather disgruntled, “What does that have to do with a framework?”

“I mean,” Liz continued calmly, “that in order to use the tool directly, you must have at least some development skills. With a framework it would be possible for testers who didn’t know the tool’s scripting language to automate test cases.”

“Hmm,” Ronald muttered sceptically, “and how would that work?”

On her sign, Jerry switched on the slides she had prepared in advance.

⁷ From Wikipedia, Expert system

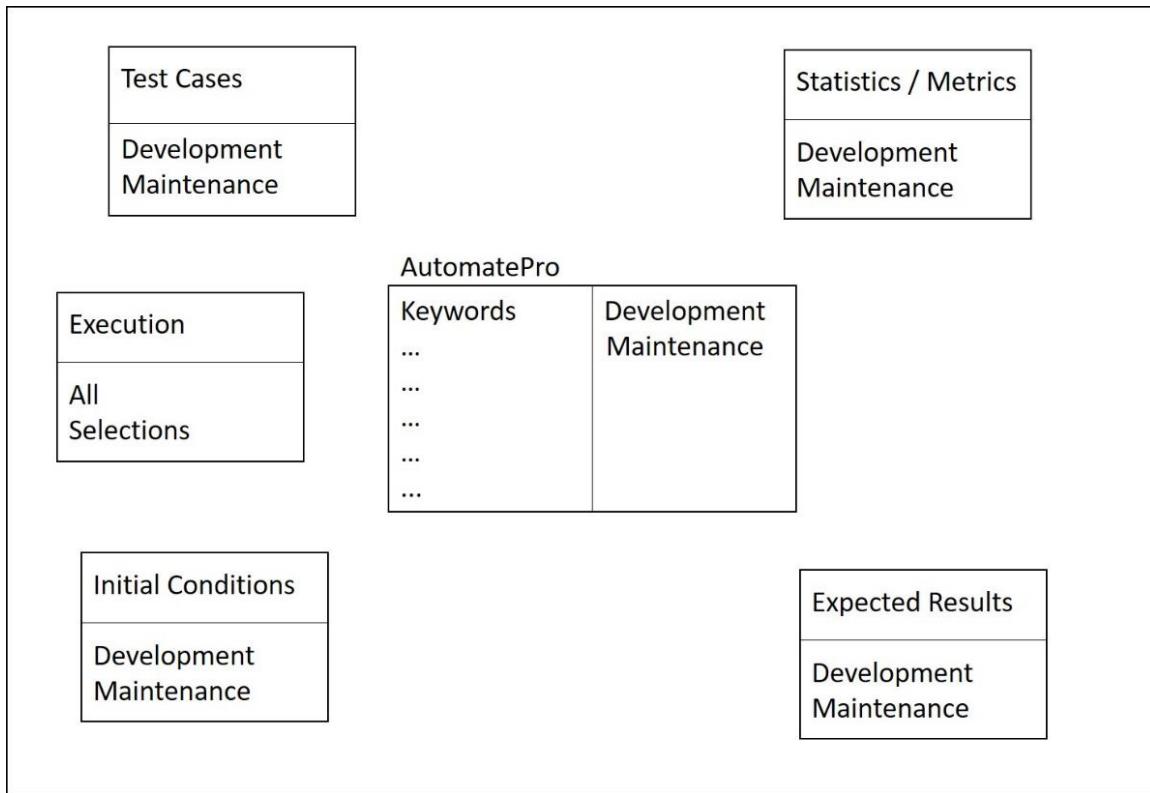


Figure 1.1.13-1

Liz went to the board and pointed to the big box in the middle of the first slide “This is the tool we are currently using, AutomatePro,” she explained, “and here are the keywords –the ones we already implemented and the future ones. With the tool we can develop them, test and maintain them”

Before she could go on, she was interrupted by Leroy. "Wait a minute," he said, "can you explain what you mean when you speak of 'Keywords'?"

"Sure," she replied, smiling. "since non-developers don't usually know what a function is, they have started to call them keywords, but it's exactly the same thing: you call it, pass it some parameters and it does something." She added with a smile, "You can call them 'functions for dummies'!"

The two developers laughed, and the testers were not sure if they should also laugh or be offended.

Still looking at Leroy, Liz added, “One of the most successful test automation design patterns is in fact KEYWORD-DRIVEN TESTING because it tells automators to build their test cases using keywords. Just as in normal code, calling functions - here keywords – forces you to be modular and ...”

Ronald interrupted her, grinning sheepishly, "Yea, I found out recently just how much work you can avoid that way."

This confession brought out understanding chuckles from the developers and Jim said, "Yes, I also still remember when I discovered that myself."

Jerry laughed and said, "Yes, we have all had similar experiences. But now we should get on with our business. Liz, please go on."

Liz then pointed to the box titled 'Test Cases' and added, "This is a tool to enable testers to write and maintain test cases using the keywords from AutomatePro. Note that at this point it's completely irrelevant how this can or will be implemented technically."

Looking at the developers she then briefly explained that the structure of automated system tests is practically identical with that of unit tests and that therefore you need a 'setup' before running the test and a 'verify' afterwards in order to check if the test has passed or failed.

Pointing to the 'Initial Conditions' box she explained how such a tool would be needed to support testers in creating and maintaining the initial conditions ('setup'). Something similar would be needed also to prepare the expected results in such a way as to be able to compare them easily with the actual test results ('verify').

And finally, she added, pointing to the 'Execution' box, "Of course testers must be able to select which tests to run and when. And this last box - Statistics and metrics - is needed in order to be able to see how testing is progressing and to discover eventual trends"

Then she halted to let them digest what she had just said. After a pause she motioned Jerry to switch to the next slide.

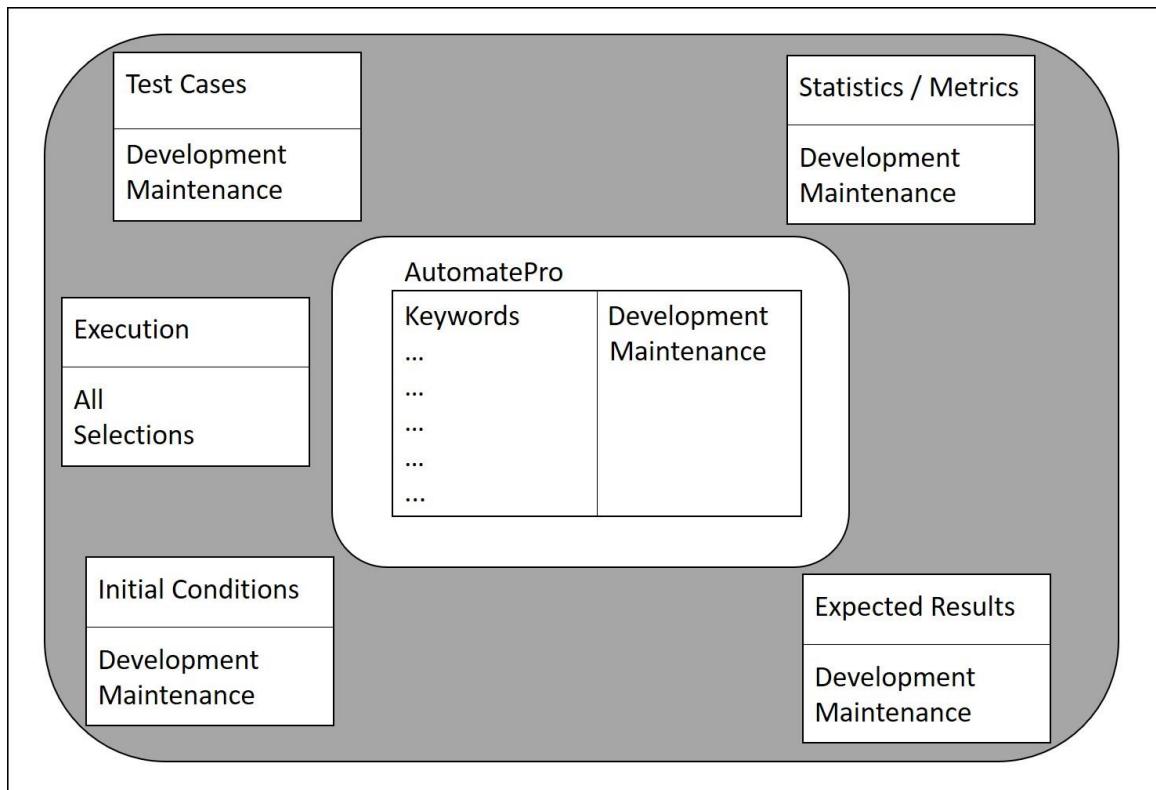


Figure 1.1.13-2

"The tools that I propose," she continued, "are not supposed to replace AutomatePro, but would interact with it and add functionalities that the tool doesn't offer. So, you see, Ronald," she added looking at him, "you have no reason to believe that I don't want to apply patterns like ABSTRACTION LEVELS or DON'T REINVENT THE WHEEL!"

Seeing that Leroy was getting ready to interrupt her, she added, looking at him "Please let me get on with my slides. I'll explain to you later about these new patterns."

"Oh no," Jill butted in, "I'll do it" and, thinking of hamburgers, smiled at Leroy.

Liz concluded, "The gray ring around AutomatePro is nothing else than the framework we are talking about. And as you can see it's not a big unwieldy and hard-to-maintain program, but just a package of different tools. I am really convinced that it would help us become much more efficient."

For a short while they were all silent. Then Tim asked, "What about documentation? We testers are supposed to develop automated tests, but how are we supposed to know about the keywords? Since I imagine that there will be more and more of them, how do we find out about them? What they do, what parameters they need, etc.?"

Liz reflected for a moment, then said, "Yes, you're absolutely right - a very good point. We should also provide a help functionality that has to be updated in sync with the keywords." And she added a new box right away entitled 'Documentation'.

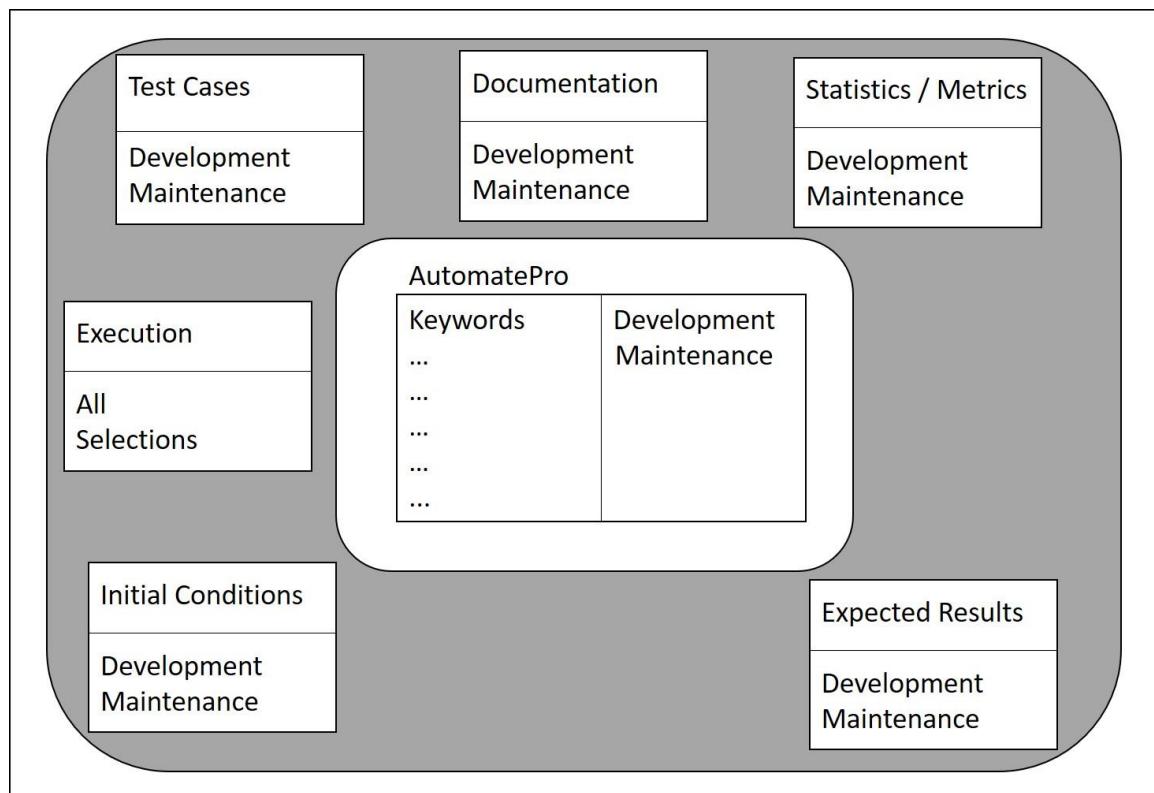


Figure 1.1.13-3

"What about configuration management?" Leroy asked, adding, "The test automation stuff should be saved with the appropriate SUT releases, don't you think?"

This time it was Jerry who agreed. "Yes, good point, that's really important. Liz, can you please add also Configuration Management to the slides? In the meantime, let's get some coffee."

When Liz joined them in the coffee kitchen they were all discussing the framework.

"The only thing that I really miss with AutomatePro," Ronald was saying, "is that there is no really complete search functionality. To find something I usually have to search with a simple text editor that at least lets me know in which keyword script I should look. I must admit that that's not really ideal!"

"What I miss," Jill interrupted, "is how the tests are doing in the long run. Paul always asks me if it's getting better or worse, so in the meantime I copy all the results to an excel sheet to be able to compare them from day to day. If the statistics tool that Liz was talking about can accomplish this automatically, it would save me a lot of work!"

After the coffee break the newly updated slide was showing on the screen, and Liz explained "I have added not only the functionalities we were talking about, but also some arrows to show which 'tools' would be driving AutomatePro and which would get some kind of information from it"

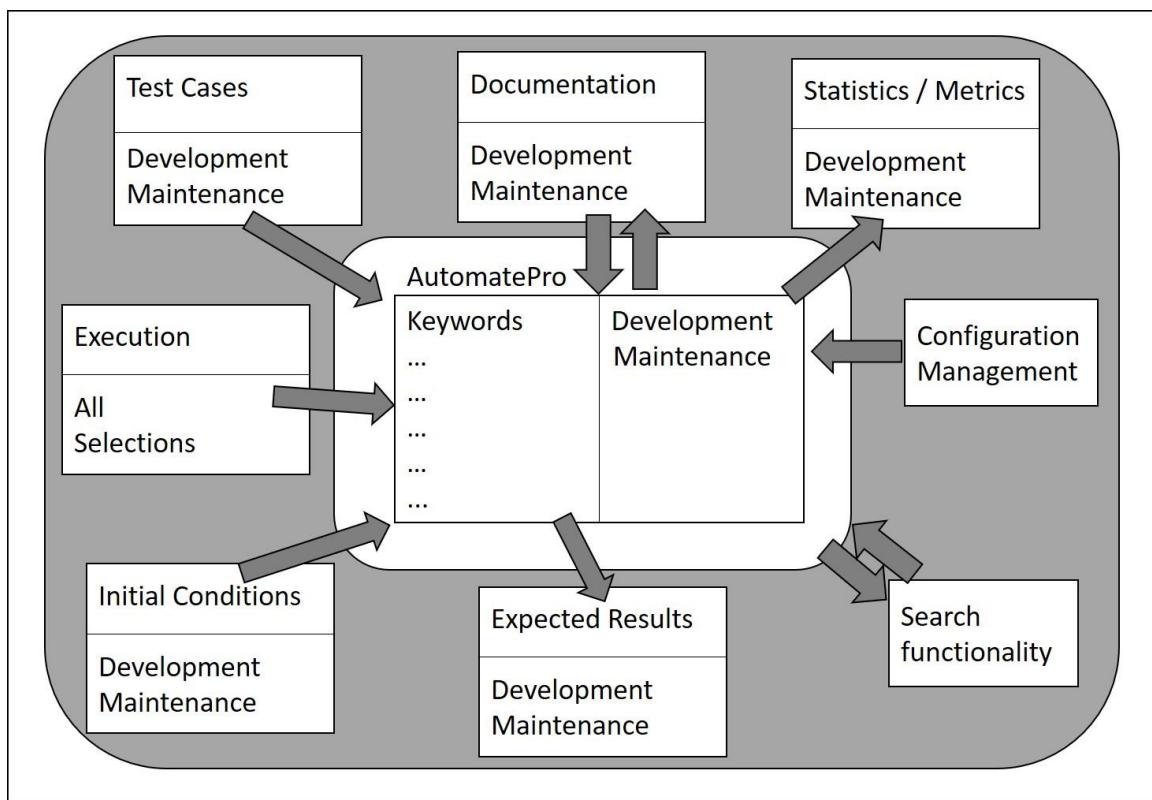


Figure 1.1.13-4

They all studied the slide and after a while both Jill and Leroy started to speak. With a smile and a nod of his head Leroy let Jill go on first. She noted “I think the arrow for the expected results points the wrong way. Aren’t we giving the expected results to be compared with the actual ones?”

“I was going to say the same thing,” Leroy said and smiled at Jill.

“You’re right” Liz replied after a short reflection, but before she was able to correct the slide, Ronald countered, “actually, I think it is right as it is, because the expected results have to first be created using AutomatePro. I would just double the arrows like you did for the Search functionality”

Liz changed the slide

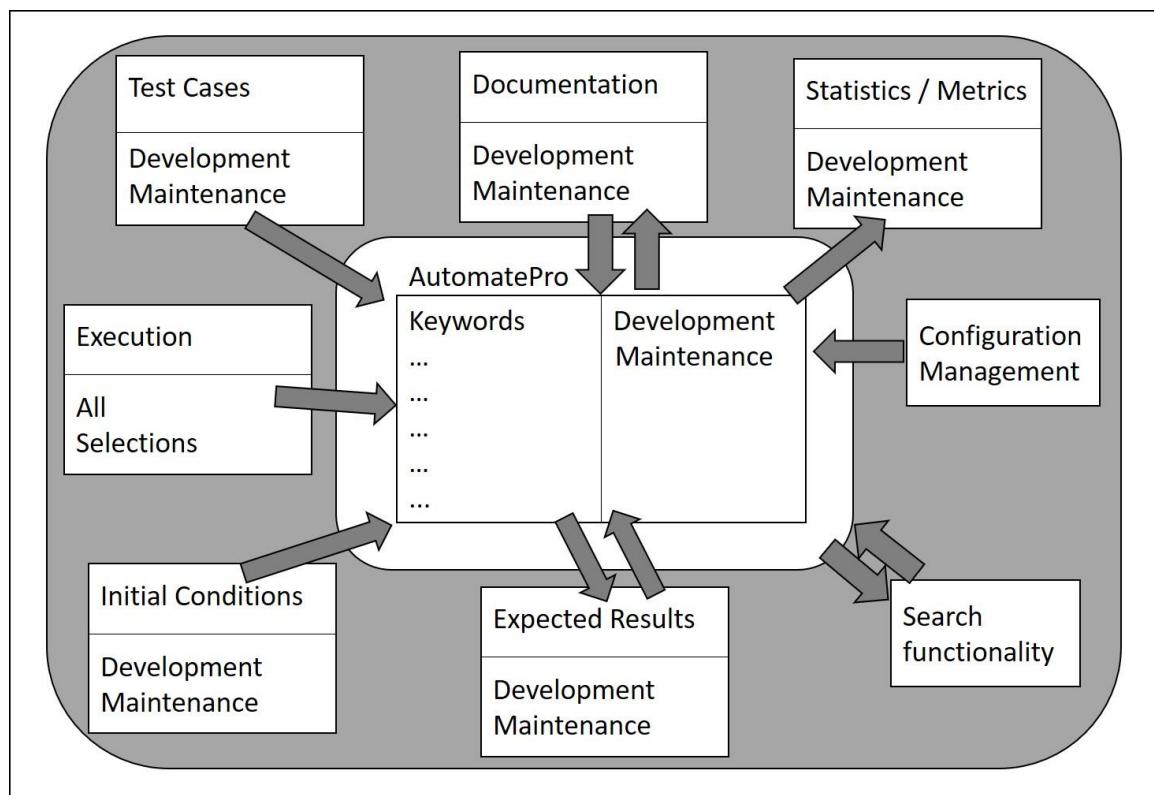


Figure 1.1.13-5

After considering the slide a while longer, Jill again had a question: “I don’t see where the results are going. Execution calls the tool, but where do the results go? You don’t need only statistics but also bug reports etc.”

“Hmm,” mused Ronald pointing to the box with the expected results “maybe we shouldn’t title this box ‘Expected Results’ but just ‘Results’.”

Liz considered this and added, while working on the slide, “Thinking it over, I think that in fact the statistics and metrics should be derived directly from the results, so I’m changing the arrow there too. Are you with me?” she asked.

Leroy approved and added, “Please, while you’re at it, correct also the arrows for Configuration Management. I think they also should point two ways”

After she had done it, he exclaimed “I think now it looks ok.”

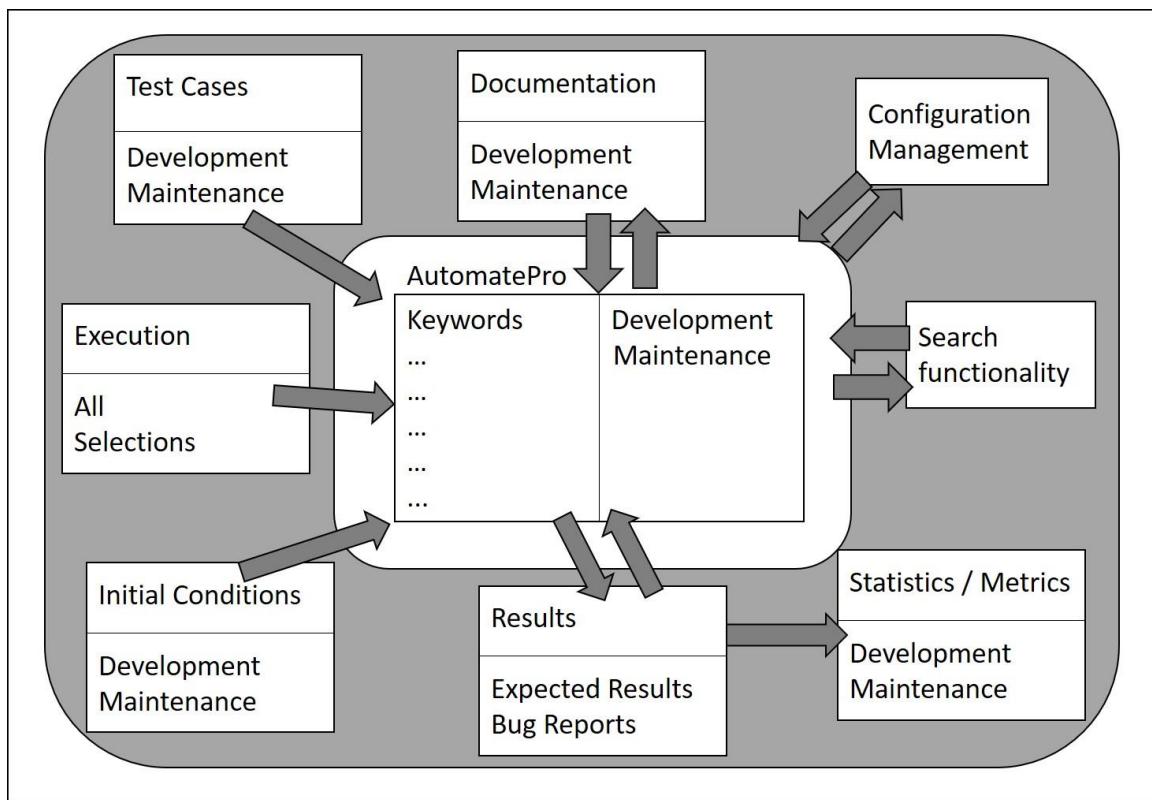


Figure 1.1.13-6

Pointing at the updated slide Jerry then asked if they had further comments.

Tim said, “Something doesn’t seem quite right here. Expected Results are here shown as part of Results, but surely that’s wrong. Expected results are part of a test case - if you don’t know what the software is supposed to be doing, then you are just playing with the system, not testing it!”

Ronald and Jill both said “Yes!” at the same time, but Leroy, Jerry and Liz looked confused.

Liz said, “But didn’t you say that you derived your expected results from the tool? So, isn’t that the same as the other test results that come from the tool?”

“No,” said Tim, “and yes. When we run a test for the first time, we look at what the application screen or a set of data is for a particular test and check that everything is ok. Then we save that screen or data as our ‘golden version’ of those results to compare against when the test is run again. So, in a way, we do get our expected results from the tool, but actually they become our ‘Expected Results’ only after we have approved the first ‘Actual Result’ from the test.”

“Yes,” said Jill, “and so the ‘approved’ result, what we call the Expected Result, becomes part of the test case, because the actual result, produced by any other test run, has to be compared to something to decide whether the test passes or fails.”

Now the penny dropped, and Liz, Leroy and Jerry understood. Liz said, “Right, let me try once more with this diagram. Is it time for a coffee break?”

Jerry brought Liz a coffee so she could work on her slide in the break. He also pointed out that she could use double-pointed arrows instead of two separate ones; for some reason, they both found this quite funny. After the break, Liz showed them this version:

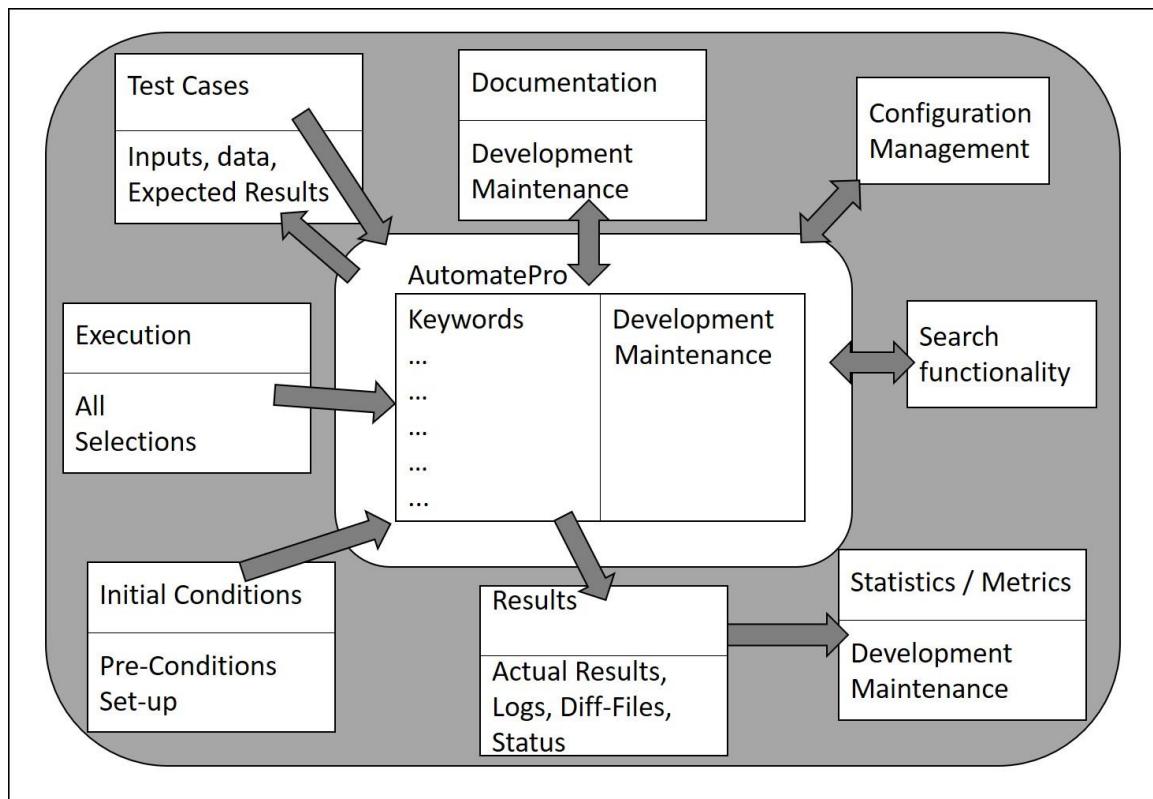


Figure 1.1.13-7

Liz explained that now the Expected results went from the tool back into the test cases, then the test cases went into the tool to run, with either all of them or a selection being executed. The Initial conditions now included any necessary pre-conditions (like an existing customer) and set-up, and the results came from the tool and included the actual results that would be compared to the expected results, plus the test logs, difference files and the test status (pass or fail), and these then would go to the Statistics / Metrics tool.

Finally, Jerry asked again if anyone had any more comments, and since this time no one answered he added, “Now that you have heard the pros and cons for this framework, what do you think?”

The first to speak up was Ronald. “I’m still not convinced” he said, “do we really need all those ‘framework-tools’? I mean they would surely be nice to have, but it would take a lot of time to implement them, and who is going to maintain them? We would have to have developers working only on the framework! Tom will never go for that!”

Now Leroy stepped in “Ronald is right! We are already understaffed now and I’m certainly not going to let one of the developers waste time playing with test automation frameworks!”

Jerry interrupted them “Wait a moment! We want to decide if we want to implement a framework, not who does it; I would discuss that later with Tom. Let’s first hear the various opinions.” Addressing Ronald he asked, “Are those your only reasons against implementing a framework?”

Ronald reflected a short while and then said “Yes, even if developing such a framework would mean making things more comfortable for us testers, I believe that in the long run it wouldn’t work. Liz has pointed out that AutomatePro is missing some important features: I can accept that, but do we have to do it ourselves? Maybe there are frameworks on the market that would give us what we need.”

“Jill, what do you think?” asked Jerry.

Jill hesitated, then said, “For myself, I would really like to have those ‘framework-tools’ and I don’t really care how we get them.”

Tim smiled and muttered to himself, “Jill, always diplomatic,” then went on louder, so that they all could hear him, “I would also like to have these tools or features, as it would make our work much more efficient, but I actually agree with Ronald that in the long run it won’t work out.”

“Why not?” asked Jerry, before Liz could butt in.

“Well, I’ve been with Digiphonia from the very beginning and, if you want me to be really honest, I can see exactly what will happen: in the beginning all is super! Then we’ll get some problems in development and everybody who can somehow code will be switched to development. It happened a year ago with Patty. She was actually an excellent tester, but she had done some coding in her previous job and before we even realized it, she was off to development. Do you really believe that that wouldn’t happen with the ‘framework-developers’? And then, nobody is left to support it and slowly, but surely it will be abandoned.”

Leroy added “Yes, and since she didn’t want to just do development, she left us shortly afterwards.”

Jerry was visibly troubled by these statements and for a moment didn’t know what to reply.

Liz was also shocked by this news. She thought that she was lucky to still be doing test automation and immediately took the opportunity to respond “If that is so, then I think that Ronald is right, and we should first see if there already is a framework or available tools that would meet our needs on the market. If there is none, then, but only then, we’ll have to swallow the bitter pill and develop one ourselves.”

In the meantime, Jerry had recomposed himself and took again the lead. “Yes, I think that’s a good idea. Still, Liz has shown us that a framework need not be monolithic, so let’s see if we can implement some of the ‘tools’ without too much

effort." Seeing that everybody agreed he continued "Leroy, Liz, let's meet again and examine what we already have."

1.1.14. Implementing Solutions

Plan out the framework,
then implement step by step
and then celebrate!

The next ‘Framework’ meeting, more than a week later, was much less formal. Just Liz, Jerry and Leroy had come together.

The week before, Leroy had sent Liz all kinds of links, so many in fact that she still hadn’t had the time to look at them all. So, she asked him, as the first thing, to explain the build and deploy framework they were currently using for the apps.

Leroy was only too happy to oblige. He had been the champion for using the open-source framework and knew it inside out. As he explained, with Liz asking a lot of detailed questions, they realized that in fact, contrary to his original opinion, it would be a very good idea to expand it to also support system test automation. In that way it would become quite easy to start the system automation runs right after the build and unit test runs.

Building on those insights, Jerry proposed that Liz and Leroy examine each ‘tool’ in the framework to find out if something similar had already been implemented internally. If not, they would first seek some external solution and if nothing was to be found they would have to decide how much effort would be needed to implement it. Once he had the ‘numbers’ Jerry could discuss the actual plan with Tom and Paul.

So, they quickly found out that some parts of the framework that Liz had wanted to develop had actually already been implemented, but currently only the developers could use them. Just a few enhancements would enable creating and maintaining test data to become much easier for non-developers to do.

Later, when Liz had told them about it, and they had realized how helpful that would be also for manual testing, Jill, Tim and Ronald couldn’t stop grinning among themselves!

The next day as she passed Paul’s office Liz found him cursing in front of his computer.

“Hi Paul, what’s wrong? Can I help you?” she asked.

“Aw, I don’t think so. Jerry has implemented the JBa(ch)-Screech and I’m busy testing this jazzed up stuff and I must set up the parameters every time and I mistype something and then I have to start again - it drives me crazy!”

Liz immediately knew she could help. “I think I can help you,” she said and continued, “There is a pattern called AUTOMATE WHAT’S NEEDED. Let me show it to you” and told him how to open the appropriate page:

AUTOMATE WHAT'S NEEDED (Process Pattern)

Automate what the developers or the testers need, even if it isn't tests!

Context

This pattern is appropriate when your automated tests will be around for a long time, but also when you just want to write one-off or disposable scripts.

Automating what isn't needed is never a good idea!

Description

Automate the tests that will give the most value. "Smoke tests" that are run every time there is any change, for example, would be good candidates for automation, as they would be run frequently and give confidence that the latest change has not destroyed everything.

When you think about test automation, the first tests that come to mind are usually regression tests that execute themselves at night or on weekends. Actually, with automation you can support testers in many other ways because even when testing manually (even with exploratory tests) there are lots of repetitive tasks that could be easily automated. You may need only a little script in Python or SQL to make a tester's life that much easier.

Implementation

Some suggestions:

- AUTOMATE GOOD TESTS: Automate only the tests that bring the most Return on Investment (ROI).
- KNOW WHEN TO STOP: Not all tests can or should be automated.
- SHARE INFORMATION: what do testers need? What could you deliver them? Start supporting them there.
- Try to get at least some developers "into the boat".

Good candidates for automation in addition to tests are:

- Complex set-ups: they can be easily automated and are great time savers for testers.
- DB-Data: Database-data can be automatically extracted or loaded for use in creating initial conditions or checking results. Such support is valued by developers and testers alike.

Potential problems

When people get "stuck in" to automation, they can get carried away with what can be done and may want to automate tests that aren't really important enough to automate.

"This does sound interesting," mused Paul after reading it. "Do you mean that you could automate some of this stuff for me?"

"Yes, I do," replied Liz and added, "actually we already have for our test automation. It's not a problem to adapt the scripts so that you can also work with them!"

Paul's eyes were shining, "Now that really would help!" he said, "when can you do it?"

"Let me go to my desk," she answered. "I bet that in half an hour you'll have it!"

When after not more than twenty minutes she was back and could show him how to use it, Paul was really grateful. "You've made my day," he said and couldn't thank her enough.

"Good automation is more than just automated tests," she answered smiling.

Presently Liz and Leroy were discussing how to implement the Execution Tool. They had already decided how to integrate the automated tests into the daily build, but Liz also wanted to give the testers the ability to select which test cases to run each time. She was just examining the Execution patterns to see if they could get some suggestions when she exclaimed, "Gotcha! Here is what I was looking for," and showed Leroy the pattern PRIORITIZE TESTS.

PRIORITIZE TESTS (Execution Pattern)

Assign each test some kind of priority in order to be able to select easily the ones that should be run

Context

Use this pattern when you may not be able to run all of the tests that you have in your automated portfolio.

Use this pattern to be selective about tests to run when something has changed.

Description

Assign each test some kind of priority, based on the criteria that are important at the time. The priority may be related to recent changes, the most critical user-facing aspects of the SUT (System Under Test), a recent change to some related system, or any other factor. The priority tests to run now may be different to the priority of the tests that you will want to run tomorrow.

You should be able to easily select subsets of tests to run, based on the priority assignment.

Implementation

- Group all the test cases that test the same functionalities of the SUT in the same test suite. In this way you can test different parts at different times or concurrently.
- Reserve a parameter on every test case for the priority and configure your tool or TEST AUTOMATION FRAMEWORK so that you can select the priority of the test cases to be run. Common values are:

- Critical
 - Important
 - Average
 - Minor
- Other categories can also be used to select which tests to run using a TEST SELECTOR, such as:
 - All the tests that failed last time.
 - The tests for a particular function.
 - Joe's tests.
 - High level smoke tests.

After reading it, Liz was convinced. "This is it," she repeated, "we have to implement both the priorities and also the TEST SELECTOR pattern!"

"I agree about priorities," said Leroy, "but I don't know about this other pattern. Let's have a look at it before we decide" and Liz clicked on the link.

TEST SELECTOR (Design Pattern)

Implement your test cases so that you can turn on various selection criteria for whether or not you include a given test in an execution run.

Context

This pattern is needed when you have a lot of automated tests, when you can no longer run all of them in the allotted time.

Description

The need for this pattern is not obvious when you first start automating, but it is an important one to take into consideration if you want to have large-scale sustainable automation.

Implementation

As part of the description or documentation of a test, include some Tags to identify this particular test in different ways. For example, as part of a Smoke Test, a regression test for a particular feature, a depth test for a function, when the test last found a bug, even the test's author. Choose your tags depending on the different ways you might like to form subsets of tests for execution.

When you DOCUMENT THE TESTWARE, this is one of the things that you will SET STANDARDS for.

When you are gathering tests for an execution run, your TEST AUTOMATION FRAMEWORK should enable you to choose the tests to run from by specifying the selector tags to include. For example, for tonight's overnight run, you may want to execute:

- The tests that failed last time they were run (to see if they are fixed correctly).
- The depth tests for the function that was most extensively changed.
- The highest priority tests from the full standard smoke test.

- Leslie's tests, as Leslie has been off sick for a few days.

Note that your framework needs to be designed to enable the tests to be selected by these Test Selectors!

The sets of tests that you choose will be determined by how you PRIORITIZE TESTS.

Potential problems

If this is not considered at the beginning of an automation effort, it is more difficult to implement, but is usually still worth doing, for the flexibility it gives to test execution.

The format of the Test Selector needs to be clearly defined and the standards for their definition and use enforced.

"Oh-oh," exclaimed Liz while reading it, "We haven't set standards for it: we completely ignored that," and she made herself a note for Ronald and the others.

Leroy was considering the problem. "I don't know if our framework can do this," he finally said. "When we start a build we always execute everything, but surely we're not the only ones to have this problem to solve. Let's call Alice, I think she did something similar in her last job."

After they explained to Alice what they needed, she replied, "Yes, I have seen something similar in my previous company, so it's definitely possible, but I need some time to think about it some more. I'll let you know as soon as possible."

Liz decided that before they could go on, she would have to ask the testers what kind of tags they would need. "Let's continue tomorrow," she said, and added, "Now it's time to go to lunch, and I'll invite Jill and Ronald to give us the information we need."

At the meeting the next day Liz explained quickly what they had done the day before and why they now needed Jill and Ronald. She showed them both patterns, PRIORITIZE TESTS and TEST SELECTOR and asked what kind of tags they would need.

After some discussion, they decided to take the tip from the pattern and settled on the following tags:

- the tests that failed last time they were run.
- the highest priority tests from the full standard smoke test.

Near the end of the meeting, Alice dropped by. She had searched for a solution and found it. It was a bit tricky, but she knew now how to proceed and was eager to start.

In each of the meetings they were discussing only one 'tool'. They would get input from the testers about what they would need and / or expect from it. Then Leroy

would look if they already had implemented something similar for the developers and Liz would search the internet to see if some solution was available. If not, they would consider how to implement it, often with some other developer who had more expertise in that particular topic. Discussing the framework with so many people made for slow progress, but everyone's contributions turned out to be really important.

Jerry dropped by quite often. With Liz he seemed to have overcome his inborn shyness with women. It was apparent that he was quite at ease with her. As for Liz, she was finding it more and more difficult to behave with him just as with everybody else, but she never noticed how different his behaviour with her was when compared to other women.

In the end, what proved to be the most difficult to implement was supporting the development of the keyword-driven test cases themselves. Liz wanted this to happen externally to AutomatePro, but at the same time the testers wanted to be able to scroll the current list in the tool in order to select and use any existing keyword. It took a lot of trial and error for their pilot implementation to finally do what they wanted it to do, and even longer to get the approval of all the testers.

Between the meetings, Liz had not only implemented more and more keywords, some completely new and some based on what Ronald, Jill and Tim had already developed, but had also continued with her lunchtime patterns lessons. After some initial scepticism, a few developers had also become steady members of her small test automation group and they had been quite helpful solving some technical problems. In this way it was more than a month later before they could give Jerry his 'numbers'. Having done quick pilots, they had even been able to implement some simple solutions and the testers were applying them already for manual testing.

Having discussed the results with Liz and Leroy, Jerry had a few meetings with Tom and Paul and then finally the implementation plan was ready. Implementation of course didn't mean that all the tools were to be developed in house; somebody still had to be responsible for providing and eventually adapting the ones that were available externally.

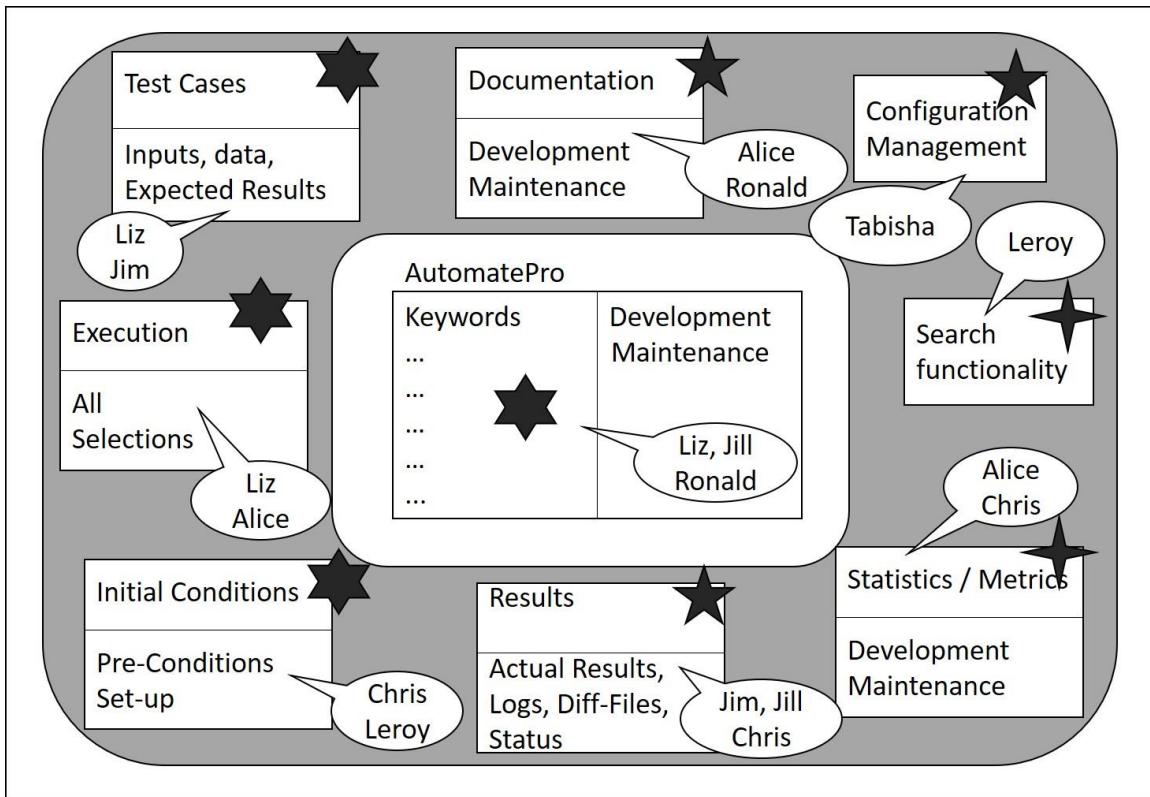


Figure 1.1.14-1

The tools marked with the six-pointed star were to be implemented first, then the ones with the five-pointed star and lastly the ones with the four-pointed star. Since, except for Liz, different people would be working on different projects, they were to be implemented in parallel. A poster with the plan was hung up in each office and every time something would be finished the priority star was to be turned green.

Finally having a plan for the work, they decided that they had earned a little celebration. Testers and developers who had worked on the 'Framework' Project were there and of course Jerry, Tom and Paul. In fact, Paul had initiated it after looking through the patterns. He had taken a special fancy for CELEBRATE SUCCESS and suggested to have a little party for every green priority star!

CELEBRATE SUCCESS (Process Pattern)

When you have reached an important milestone, it's celebration time.

Description

When you have reached a milestone in your automation, it's time to celebrate. A milestone might be automating your first test suite, establishing a level of coverage, or a number of unattended automated tests being run.

Implementation

Invite the team, the testers and all the people that supported you (for instance that database expert that helped with some special SQL statements).

Depending on what you celebrate, you can offer:

- Special treats for coffee time.
- Drinks.
- Pizzas.
- After work party.
- Dinner at a restaurant.
- Other entertainment.
- Small presents.
- A bonus.

Celebrations don't need to be large or expensive - put some thought into what would be appreciated - sometimes just some recognition of the effort that has gone into the automation is what is needed.

Recommendations

Invite your managers to every celebration. Show what you have already accomplished and tell them that you appreciate the support they have given you already. Share your plans for the future of the automation. (This can be a subtle way to remind them that you need their continuing support for further progress.)

Suggestions:

- calculate how much time has been saved by running automated instead of manual tests.
- show how test coverage has grown due to test automation.
- explain how your test automation solution works.
- tell about a regression defect that has been found by the automated tests before delivery of the new release.

1.1.15. Management Problems

Management is key
for automation success
They must support it

After all these preparations, everybody went back to work, but with slightly different tasks so that the framework could be implemented according to the plan. So, for instance Leroy would work for 30% of his time on the framework and Alice 20%. For Liz this meant 50% automating test cases and 50% building new keywords or implementing framework tools. For the testers it was similar, but they could now do more manual testing in a shorter time, as they could already use some or parts of the framework tools.

A couple of weeks later, Tim went to Liz and showed her the site for a testing conference. "Liz," he said "they started a call for papers for the next conference which will be in approximately six months. Why don't you go and talk about how you turned over our stalled automation?"

Liz thought about it a moment then answered "Yes, why not? I'll think of something. Thanks for your tip!"

And in fact, she submitted a short abstract, 'How we restarted our stalled test automation using the Test Automation Patterns Wiki' and was later very proud to have been selected.

Some weeks later Tom, Jerry and Paul were discussing together a request from a big media company. They wanted Digiphonia to develop exclusively for them a film-music Screech-app. They were willing to pay a very substantial sum, but they wanted the app in 3 months.

"To develop something like that in three months...I don't think that's possible" Jerry was saying.

"Even if we drop everything else?" countered Tom.

"Even if we drop everything else." confirmed Jerry and continued, "The problem is that film music is very different from movie to movie. You can't do the same music for a love story and for an action film. Also, the music is very dependent on what scene is currently playing. Just to thinking about how to solve the problem, I'd need more than three months!"

Tom looked really disappointed. "I don't want to lose this chance," he affirmed, "I'll try to negotiate for more time," he said and added, looking at Jerry, "What can you deliver and in how much time?"

Jerry reflected a long while. Finally, he said "First of all, I would start with only one type of film, for instance an action movie. Then we should define at the most three types of scene, let's say a chase, an ambush and a dialog. Then I need a big library

of that kind of film music both as audio files and as scores and of course the movies to go with it. And it would help to have somebody else set up the statistical calculations..."

Paul interrupted him "Yes, and I would also need somebody to help with testing. I'm not so sure I would know if some kind of generated music fits any one movie scene."

Tom had listened attentively and asked, "Suppose you get that all, how much time would you need?"

"Not less than six months" answered Jerry immediately "rather more...and we would really have to drop everything else."

Tom sighed "I'll talk to them. Does it matter what kind of film you could start with? I think if we can deliver the app for the next movie they are planning, I could convince them to start with just that. I could tell them that we want to do a pilot to be sure to deliver just what they need."

Jerry grinned "You've always been a good salesman, Tom!"

It took quite a bit to convince the media company that their expectations were unrealistic and that they would have to start just doing a pilot⁸, but finally Tom convinced them. As soon as the contract was signed, Tom and Jerry assembled all employees in the big conference room.

"I have some very good news," Tom started. "We have a really substantial order for a Film-Music Screech-app and I want you to work with high priority on that. Drop all other tasks. We are also going to hire a specialist in film music and another expert in statistics to take some of the burden off Jerry and Paul. We have already published ads and if anyone of you knows of somebody that fits one of those roles, please let us know. We will pay you an extra bonus."

The room was at first completely silent and then exploded in dozens of different discussions.

Liz was the first to speak. "What about the test automation?" she asked. "If we just stop now, all the achievements that we've had until now will just disappear!"

Jerry replied "We're just going to run with what we have now, and for the time being we'll do no new stuff. We won't need the test automation for the new apps right away anyhow."

Tom added, "We'll need every developer and also anyone with some coding experience will also be added to the development team."

⁸ Note that some issues and patterns come up not only in test automation. Here *UNREALISTIC EXPECTATIONS* (Issue) and *DO A PILOT* (Pattern)

Liz shuddered, thinking of what Tim had told them about the tester Patty, but before she could answer Ronald butted in.

"What about the apps that we were testing?" he asked, "We're not through with testing. They can't be released yet!"

Paul answered "You're right, but they'll have to wait. Of course, if you have time to spare, you can continue to work on them, but only if you have spare time."

All work on the test automation framework stopped. No new test cases were to be automated.

Liz was now part of development, even sitting with them, and had only about 10% of her time to start and check the existing automated tests. Every time some test didn't run because of changes to the System under Test (SUT) she was supposed to simply remove the test. Such tests would be corrected later. She had to work so hard to catch up with her developer tasks that she also had to stop with the lunch lessons. She was getting more and more frustrated. She worked conscientiously, but not being allowed to progress further with the test automation, she had lost the inner drive that she had shown before. When she had done her eight hours she would go home even if that meant leaving things unfinished.

She still played her bass (she had even started practicing on a Bach fugue) and never missed a rehearsal, but even there she was quieter and more reserved than before. Once Tigg asked her about it, but Liz claimed to be perfectly ok. From then on, she tried to give the impression that nothing was bothering her, but her friends noticed just the same that something was wrong, very wrong: she had stopped saying 'awesome'!

The only bright spot was that, by going to lunch with the other developers, she could see Jerry every day! She had gradually fallen in love with him but was convinced that her love could never be reciprocated: an intelligent man like Jerry and owner of a very successful start-up could have any woman he wanted and was certainly not interested in just a test automator! She was thus very careful to never be other than friendly towards him and that in a decidedly neutral way. In fact, nobody noticed that her interest was of a more personal kind.

Jerry for his part was practically closed up the whole day in his office trying to find the best way to implement the app. On one hand he loved the challenge, but on some days, he would have liked to throw in the towel and just disappear! For him too, being with Liz more often 'made his days', but he was also hiding his feelings for her, not having seen any special reaction from her: she was friendly with him, but she was friendly with everybody. Just like Liz, he was sure that there was no way such an attractive and intelligent woman could 'fall' for him, 'Big Ears'! Tom and even Paul were now convinced to have been wrong in suspecting that he had special feelings for her.

The time came for Liz to prepare her talk for the conference and she was really down. She had been so proud to be able to tell the testing community about her success story but now she felt as if it was all just one big lie! Still her talk must match the abstract, so she had to do it as if everything was still fine.

At the conference, Liz's talk was a big success. When she was finished, lots of people came up with questions and two guys even offered her a job! "You are exactly what we need to revive our automation," the younger one said in good English, but with a decidedly German accent. "We work for a big international bank in Frankfurt and we have similar problems to what you described here. We would give you the best conditions."

The older man continued, "You would not only get a good salary," and he offered a really substantial sum, "but in Germany we also have six weeks of vacation and the company would help you find an apartment or a house and would of course also pay all your moving expenses!"

Liz was quite unprepared for such an 'attack' and spluttered "Sorry, but I can't speak German! How would I be able to work there?"

"Oh," replied the older man, "That's not a problem. First of all, our employees are quite good in English, and secondly, we can send you, at our expense of course, to an intensive German course. I'm sure that in a short time you will become quite fluent!"

Liz was overwhelmed. She said she would think it over and give them an answer in a week.

On the flight home she just could not make up her mind. At some point she realized that she had to ask for help and that thought lighted the lamp for her: she remembered that there was a pattern for it, ASK FOR HELP. "I wonder," she thought, "if it's also applicable in my case!" As soon as she got home, still in the middle of the night, she opened her notebook and looked up the pattern.

ASK FOR HELP (Process Pattern)

Ask for help instead of wasting time trying to do everything yourself.

Description

Ask for help when you need it. Recognise when you have got "stuck" and can't get any further on your own.

Implementation

Find out who has the expertise you are missing and ask for help. Do try to work on your own, but don't waste time. If somebody can help you to quickly solve some problem, then definitely ask.

Help may be available from other people in your organisation, but also from web sites and discussion groups and forums.

To encourage people to help you, you can use "gamification" (see experience entry from Kristoffer Nordström)

Recommendations

| Don't be afraid to ask for help: most people actually enjoy helping.

"Hmm," she thought, "I can definitely apply it. I just have to decide, who is 'the expert' in this case"

Sitting on her couch she considered, "I would like to ask Jill or Ronald or Leroy, but I can't ask anybody at Digiphonia, as I don't want them to even suspect that I might leave. Also, Jerry would know best, but I couldn't stand it if he advised me to go!" After a long while she finally had the right intuition "I know," she told herself, "Jack and Tigg are the 'experts' I need."

So, on Monday, she called Tigg and asked if she could visit her, she had to speak to somebody.

"Of course," replied Tigg "you're lucky that today I'm not on duty, I'll cook something, and you can tell me what's going on."

Later at the table, eating Tigg's 'special', -Chicken Hungarian style with Nockerli-, Liz told Tigg how frustrated she had become. She concluded "I wanted to do test automation and now I'm just another developer and who knows for how long."

"So, what do you want to do? Find another job? Remember that nowadays jobs are not so easy to find"

"Ah, but I do already have an offer," replied Liz beaming "in Germany!"

"What?" Tigg jumped up and said, "Are you crazy?"

Liz laughed "Yes, that was what I thought too when they offered it to me." And she went on to describe the conditions. Finally, she said, "I am tempted, first of all because I could get away from here, but also because the job is really challenging. It's a big international bank and I could use all my prior experience to help them."

Tigg had turned pensive. "Hmm," she said, "it is a difficult decision. Still, since it seems that here you are just treading water, I think you should accept and go. After all, you didn't need all that much time at Digiphonia to turn their test automation around, so you can always come back afterwards, and you would have good references on your CV."

But then she said, "But what about music? The band would really miss you - and so would I!"

Liz smiled "I was wondering when you would ask. Lately I have been playing a lot of Bach and Bach was from Germany, so I believe that I will find a band or a small orchestra to play with."

Hearing about Bach, Tigg could only shake her head.

A day later, Liz went over to Jack's. After hearing the conditions, he too advised her to accept. It was great to see Jack and Deirdre again, and Deirdre was now very tired (and big) as the baby was due soon.

Liz had already been tempted, but after both of her best friends had also advised her to go, she accepted the offer. As soon as she had the confirmation, she went to Tom and gave notice. He hadn't expected it and tried to change her mind. Finally, he asked Jerry to speak to her.

Jerry, now definitely convinced that she would never care for him, tried to put himself in her place and, thinking only on what would be best for her, didn't even try to change her mind, but instead showed himself sympathetic to her wish to go. For Liz this was the final proof that Jerry was not and never would be interested in her, and that only confirmed her wish to leave. Tom even called his Dad to try to change her mind, but to no avail. Two weeks later she had passed all her tasks over to Ronald, Jill, Leroy and the others and was gone.

With her gone, test automation at Digiphonia was again more or less dead. Tim remembered a pattern he had learned about in the tutorial, TEST AUTOMATION OWNER and talking with Ronald, he argued that Liz had been the one and that without her no one really cared any more. Ronald had been intrigued and once at home looked it up.

TEST AUTOMATION OWNER (Management Pattern)

Appoint an owner for the test automation effort. If there is already a "champion" give him or her public support.

Context

This pattern is already necessary for individual automation efforts, but is especially important for long lasting success by larger automation projects

Description

The "test automation owner" is not necessarily the project leader, but he or she must be its "champion". The owner, once test automation has been established, controls that it stays "healthy" and keeps an eye on new tools, techniques or processes in order to improve it.

Implementation

The most important patterns for the automation owner are:

- LOOK AHEAD: keep in touch with the tester, automation and development community in order to stay informed about new tools, methods etc.
- LOOK FOR TROUBLE: watch out for possible problems in order to solve them before they become unmanageable.

Other useful patterns:

- CELEBRATE SUCCESS to keep the high motivation level in test automation.
- SHARE INFORMATION with testers, developers and management.

- MANAGEMENT SUPPORT will be needed at all times.

Potential problems

The biggest danger is that the test automation owner decides to leave the company without adequately sharing their knowledge. It is important to also have a DEPUTY. The test automation owner needs support from management if they are trying to implement changes to the automation.

"Hmm," he told one of his cats as it sprang on his keyboard, "Tim is right. If I hadn't seen it, I wouldn't have believed how easy it is to disrupt a well-going test automation effort!"

1.1.16. New company, new Issues

Don't work together,
don't share knowledge with others:
this issue is known!

Liz was slowly getting accustomed to life in Germany. Busses and trains were always on time (even if her German colleagues found it terrible when they were just a couple of minutes late), shops stayed closed on Sundays, the weather was at times quite miserable and there were stretches on the 'Autobahn' where you could drive as fast as your car would go.

She was feeling lonely, but she felt that it was helping heal her heart. She didn't want to forget Jerry, she still cherished her love, but she knew that she had to move on.

Making friends was not on her radar yet and it would have been difficult anyway. Her new colleagues were friendly but nothing more, and with her limited German she could barely handle even simple conversations.

Recently she had also found a small orchestra and was going to their rehearsals. She still hadn't been allowed to play in a concert, but she knew it was just a matter of time.

The bank was structured very hierarchically so she never even saw the 'big bosses', but her automator colleagues were mostly quite nice. In the beginning, they had been somewhat awed by her, the 'Test Automation Guru' from the conference, but they had quickly noticed her friendly bearing and had become quite 'normal'. They also valued that she really made an effort to learn to speak German and didn't expect everybody to be able to speak perfect English to her. The job was even more challenging than expected because some of the testers and automators were not willing to change anything and so they developed all kinds of delaying tactics. Sometimes Liz thought that they should have employed a politician and not an automator!

She still hadn't introduced the patterns or the diagnostic feature of the test automation wiki. Instead she had spent her first weeks just trying to understand what was going on (or not). In the company there were five autonomous automation teams:

1. The first team worked on the web services for the customer's portal. Their application had an important interface to the legacy system and that made automation difficult.
2. The second team had a completely different focus. They had to integrate an external application for reporting to the German Federal Financial Supervisory Authority (BaFin) with their internal accounting systems. Every time they got a new release they had to retest their interfaces and since new

reporting laws were coming out more and more frequently they really depended on automation.

3. The third team was just beginning with test automation.
4. Team No. 4 already had a long-established test automation project, but they were using an outdated tool and now had to switch to a more modern one. The task was scary because nobody had ever cared about tool independence and they realized now that they had to start over from scratch.
5. The fifth team was currently doing quite well, but they wanted to do a lot of refactoring and found no support from management for the time needed to do the work.

The more she learned, the more she became convinced that the issue here was *LOCALISED REGIMES*. To be sure, she had looked it up and found her intuition confirmed:

LOCALISED REGIMES (Management Issue)

Tool use or testware architecture is different from team to team. This issue is closely related to the failure pattern FRAMEWORK COMPETITION from Michael Stahl

Examples

- "everyone will do the sensible thing": most will do something sensible, but different.
- "use the tool however it best suits you": ignores cost of learning how best to automate.
- effort is wasted by repeatedly solving the same problem in different ways.
- no re-use between teams.
- multiple learning curves.

Questions

Is there an overall strategy for automation?

Is there any person charged with coordinating automation for the company / enterprise?

Resolving Patterns

Most recommended:

- **DESIGN FOR REUSE**: Design reusable testware.
- **DON'T REINVENT THE WHEEL**: Use available know-how, tools and processes whenever possible.
- **SET STANDARDS**: Set and follow standards for the automation artefacts.**TEST AUTOMATION OWNER**: Appoint an owner for the test automation effort.

Other useful patterns:

- GET TRAINING: Plan to get training for all those involved in the test automation project.

SHARE INFORMATION: Ask and give information to managers, developers, other testers and customers.

Since she wanted the teams to discover this by themselves, she tried first to apply the diagnostic from the point of view of each team hoping in this way to prove to all of them that, to begin with, *LOCALISED REGIMES* was their main issue. She found this didn't work out as she had expected, because any team, from their point of view, would always chose something different from the others. Even changing the selections again and again didn't help.

"How can this be," she thought. "Until now I have always found everything in the wiki! And they did put the issue in. Why can't I reach it with the diagnostic?" At some point she landed again on the very first diagnostic page and there, at the foot of the page she read:

(Let us know if there is a question we should be asking at any point that would be more relevant to you than the ones we already have - thanks.)

"Ah" she thought "They didn't notice that it was missing, but they say they welcome contributions, so I will suggest this issue!"

She considered what to add and decided that the best thing would be to put this notice at the end of the very first diagnostic page:

"If in your company there are different teams working with automation, let each do the diagnostic first by themselves. If the issues they wind up with differ substantially you should then look up the issue *LOCALISED REGIMES* because that's a sign that each team is working completely independently from the others, there is no sharing of know-how, methods, standards etc."

She was delighted when the organisers of the wiki thanked her for her suggesting and implemented it straight away.

"That was very satisfying" she thought after seeing the new content and she was really proud to see her name in the acknowledgements page.

"Now is the time to introduce the wiki and the diagnostic" she finally thought and set up a meeting with the leaders of the test automation teams and their most senior automators.

At the meeting, she explained first how issues and patterns had been collected in the wiki and then went on to present the diagnostic asking everybody present to help decide what answer to pick. We have already seen it, but here once more the first batch of diagnostic questions:

If you are not satisfied with your current automation, what describes the most pressing problem you have to tackle at the moment?

1. Lack of support: from management, testers, developers etc.
2. Lack of resources: staff, software, hardware, time etc.
3. Lack of direction: what to automate, which automation architecture to implement etc.
4. Lack of specific knowledge: how to test the Software Under Test (SUT), use a tool, write maintainable automation etc.
5. Management expectations for automation not met (Return On Investment (ROI), behind schedule, etc.)
6. Expectations for automated test execution not met (scripts unreliable or too slow, Tests cannot run unattended, etc.)
7. Maintenance expectations not met (undocumented data or scripts, no revision control, etc.)

The discussion about how to go on from there went on for hours! There were five teams and, as she had expected, each team picked a different answer!

- The team that worked on the web services for the customer's portal had chosen No. 3 - Lack of direction.
- The team that was just beginning with test automation had chosen No. 4 - Lack of specific knowledge
- The team that was working with the BaFin selected No. 6 - Expectations for automated test execution not met.
- The team that wanted to do refactoring selected No. 1 - Lack of support
- The team that had to change tool had selected No. 7 - Maintenance expectations not met.

Now Liz had them where she wanted them, and she showed them the first diagnostic page with the note that had been recently added. Then she jumped to the page for *LOCALISED REGIMES*.

After reading it, there was at first just silence. Then the different people looked at each other abashed as if they had just been caught with their hands in the cookie jar!

Liz was careful not to look smug and decided that this would be enough for one day. In the next meetings she would then help them produce a plan to set up a process where know-how, tools and so on could be shared more widely.

How was Digiphonia doing in the mean time? Jerry had managed to deliver the algorithm for the Film-Screech App almost on time and developers and testers had worked literally day and night to implement it. When finally, everyone could return to his or her normal rhythm, the testers started to complain louder and louder that their test automation was getting nowhere. Ronald had printed the relevant issues from the Test Automation Patterns Wiki and taped them on every door: on Paul's door one could read about *STALLED AUTOMATION*. Tom had become an expert about *UNREALISTIC EXPECTATIONS* and Jerry about *KNOW HOW LEAKAGE*.

A new automator had to be found. They had to interview quite a lot of candidates before finding one that met their expectations (both Ronald and Paul admitted that they had been spoiled by Liz...), but finally things started to get better again. Ronald took his job as ‘watch dog’ very seriously and he made sure that the standards were kept and even improved. The new automator, Bob, was quite good and had already known about the wiki, but he had to hear again and again how wonderful Liz had been. The one praising her most was...Jerry!

Later that year Bob went to a big test conference (he had applied as a speaker already before coming to Digiphonia) and there, after he finished his talk, he was complimented by none other than Liz! She was there with three of her new German colleagues and each was following a different track in order to get the most information.

“You are -The Liz?” Bob exclaimed surprised “Jerry is telling anybody who wants to hear how intelligent, understanding, charming and just wonderful you are, an angel! I thought you must have wings!”

“He said that?” Liz whispered. “Jerry said that?” she repeated, blushing. Then she regained control and with a forced smile added “I’m sure he exaggerates, I’m only a test automator like you are. But I’m curious, how is Digiphonia doing now?”

Bob had noticed her reaction and understood that her question actually meant “How is Jerry doing?” and so related how Jerry had been able to implement the Film-Screech App, whereupon Liz butted in saying, “I never expected anything else! Jerry is just the best,” and added giggling “and he’s so cute with his big red ears!”

Now Bob had already been wondering why Jerry would praise her so much and when she did the same, the bell rang. Before leaving, he innocently asked her if he could have her card in case he wanted to ask her something and she handed it over without a second thought.

Back at Digiphonia Bob had to tell what he had learned at the conference and of course he mentioned meeting Liz. When Jerry later asked about her he cited her word-for-word and handed him her card and was very pleased to see how Jerry’s eyes sparkled and his ears turned a bright red!

And did Liz and Jerry meet again? Will they live happily ever after? Will this story book have a sequel? Perhaps there isn’t a Pattern for everything!

2. Part 2

2.1. Mind Maps

The following mind maps give an overview of issues and patterns and some of the relationships between them.

Note that in the wiki both the issues and the patterns are color coded:

- Process: black
- Management: magenta
- Design: blue
- Execution: green

Here we will mark the issues or patterns with the leading character of the type:

- Process: P
- Management: M
- Design: D
- Execution: E

2.1.1. General overview

Test Automation Issues

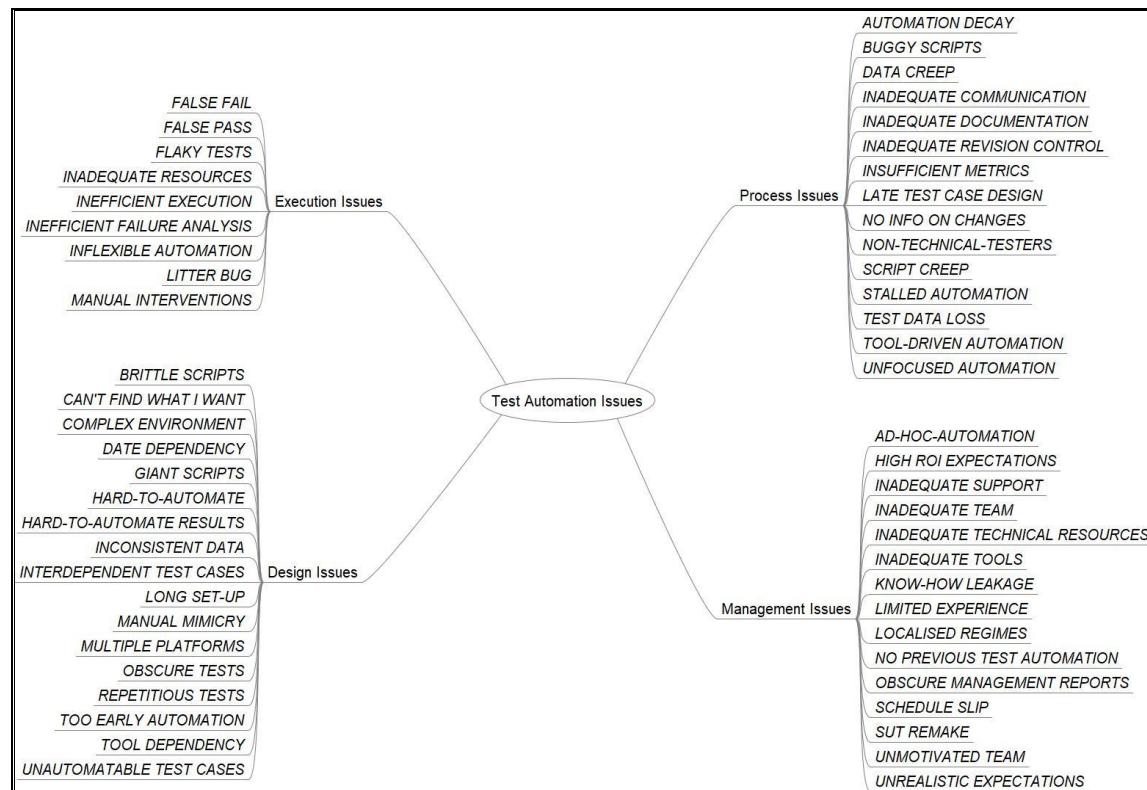


Figure 2.1.1-1- Test Automation Issues

Test Automation Patterns

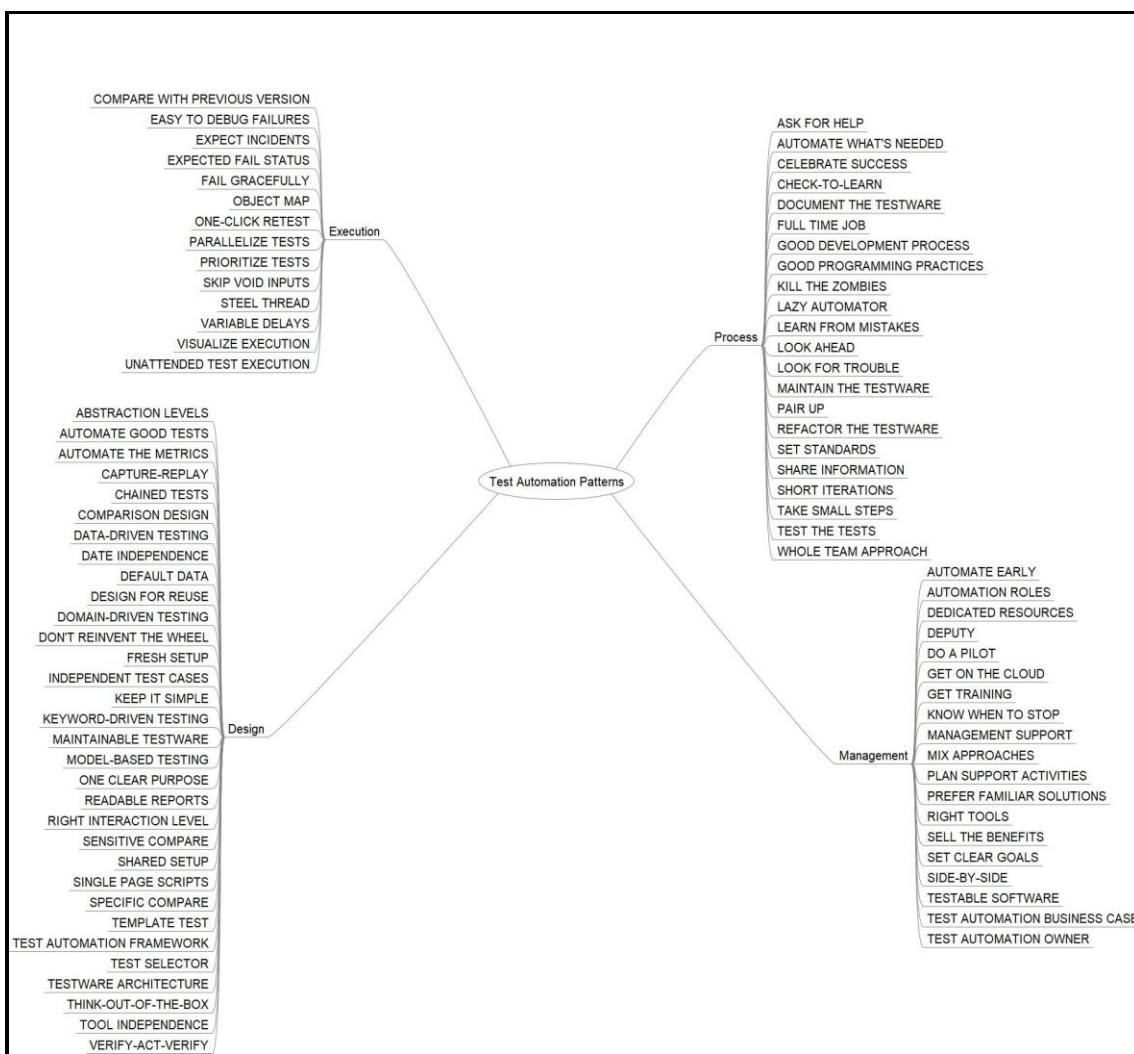


Figure 2.1.1-2 - Test Automation Patterns

2.1.2. Issues by category

In this section, the issues from each category are shown with the most recommended patterns and the other useful patterns, as shown in the issue's description.

Process Issues

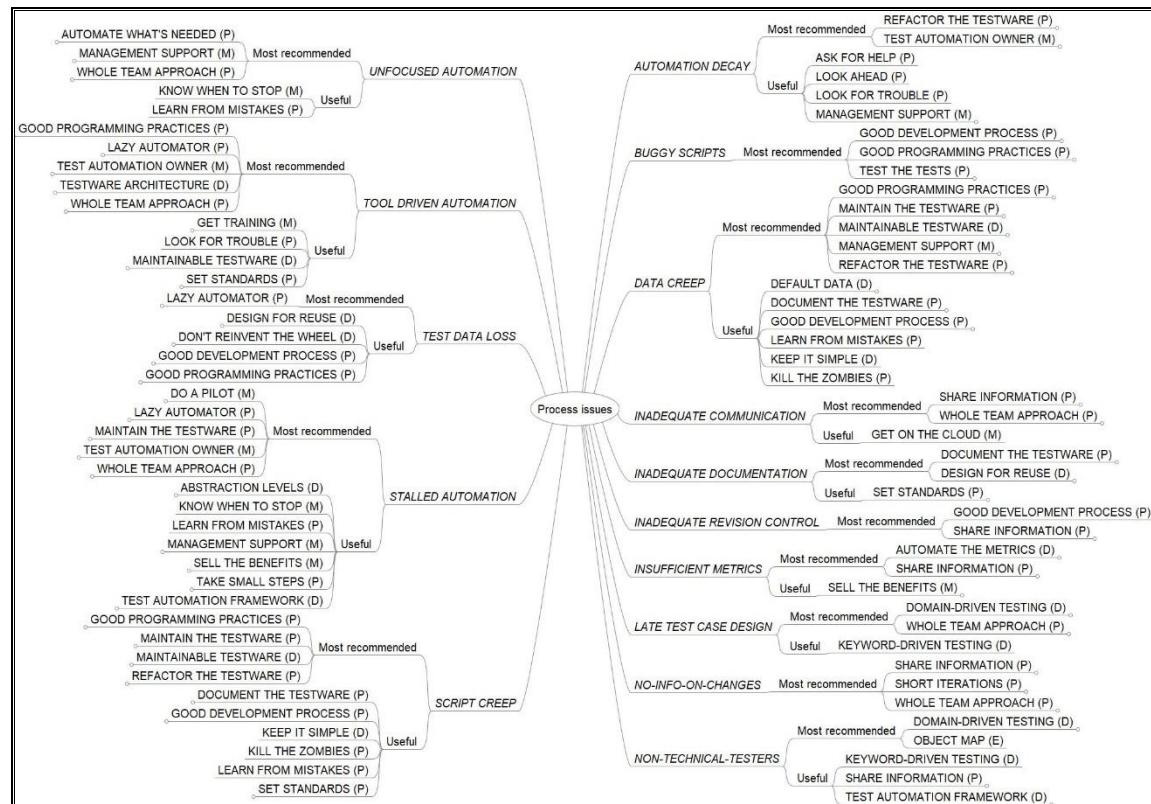


Figure 2.1.2-1 - Process Issues

Management Issues

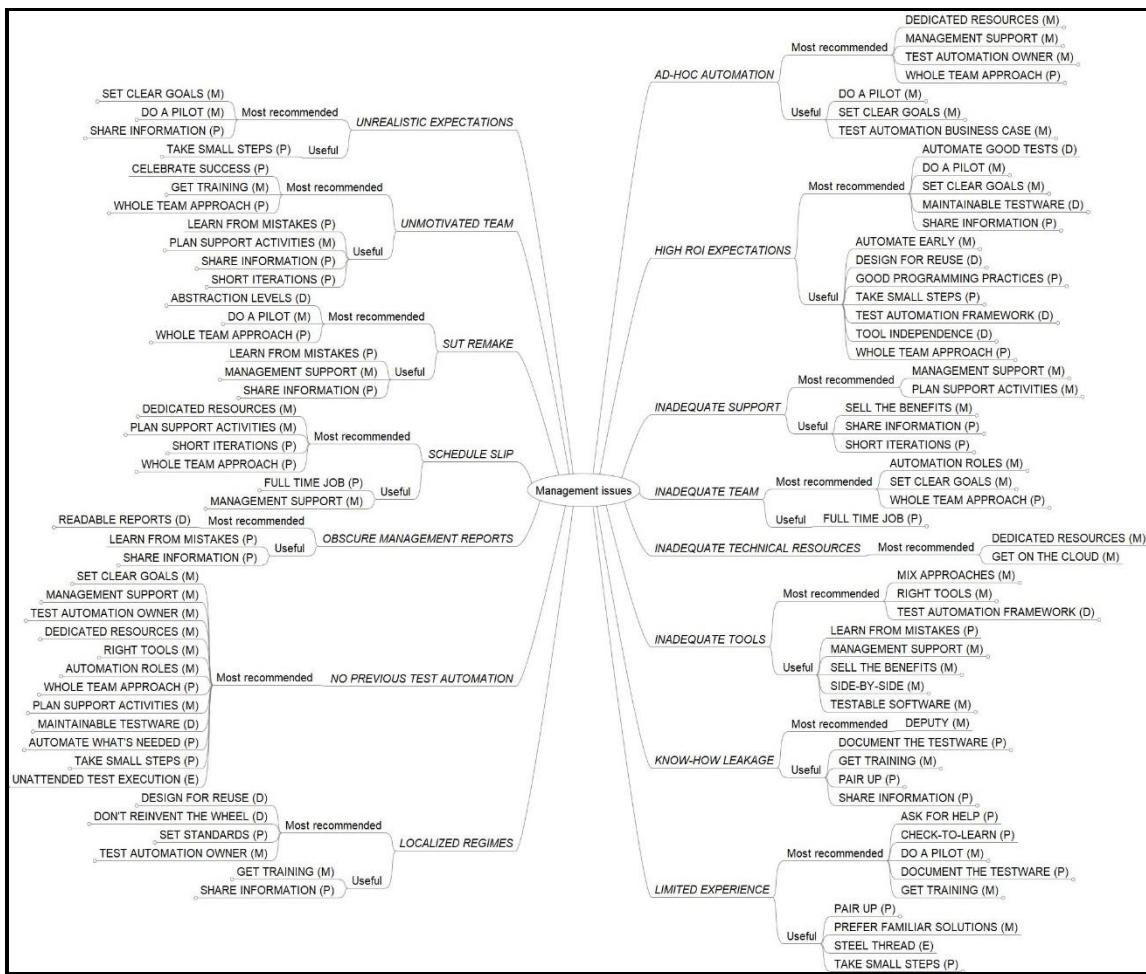


Figure 2.1.2-2- Management Issues

Design Issues

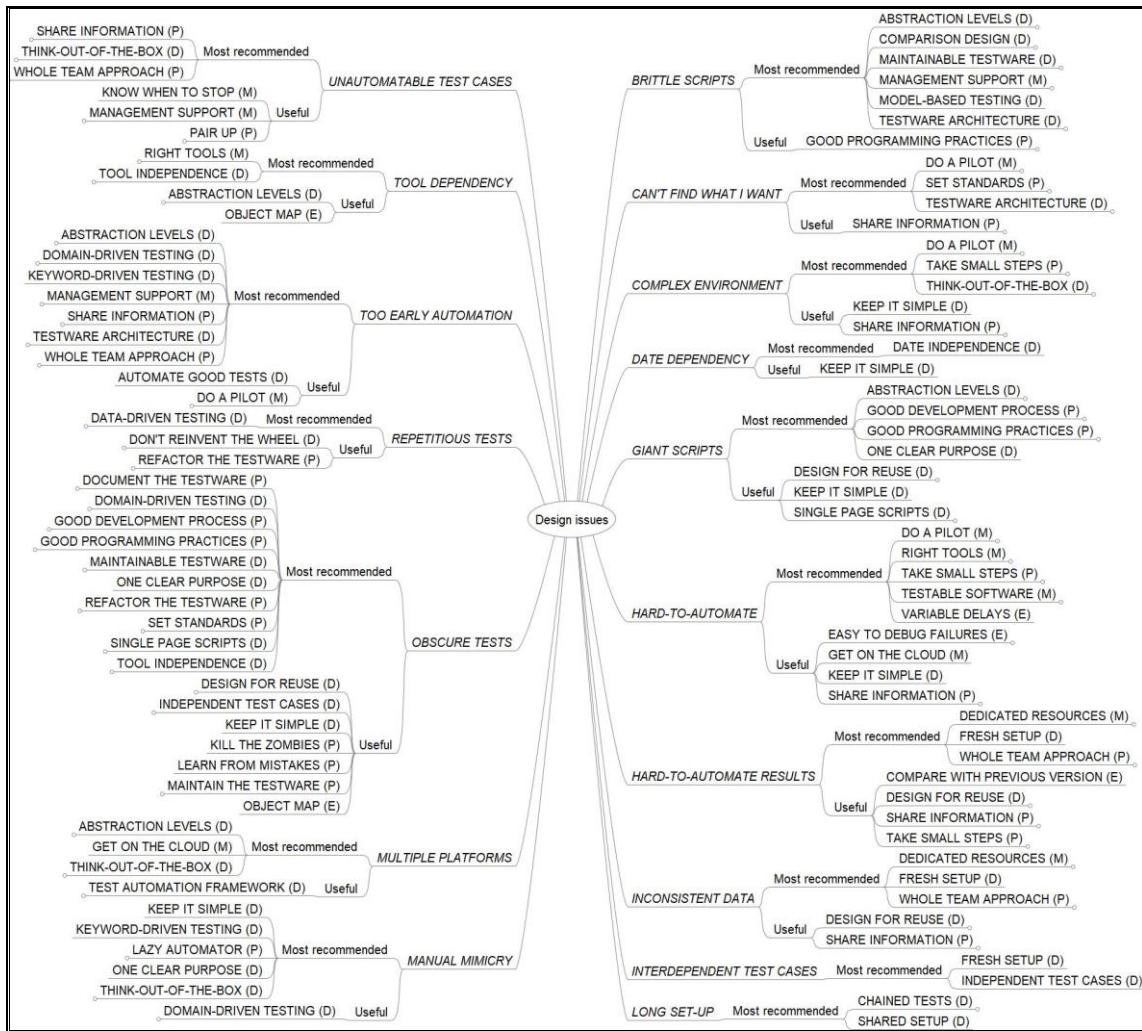


Figure 2.1.2-3- Design Issues

Execution Issues

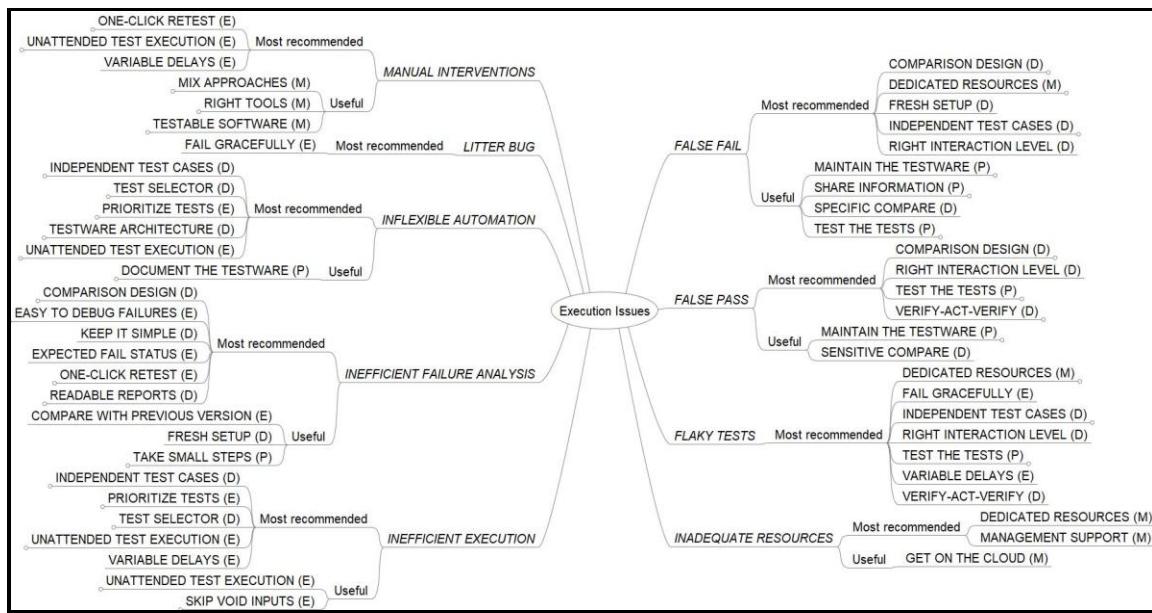


Figure 2.1.2-4 - Execution Issues

2.1.3. Patterns by category

In this section, the patterns from each category are shown with some of the other patterns that are referenced to help implement the pattern, as shown in the description of the pattern.

Process Patterns

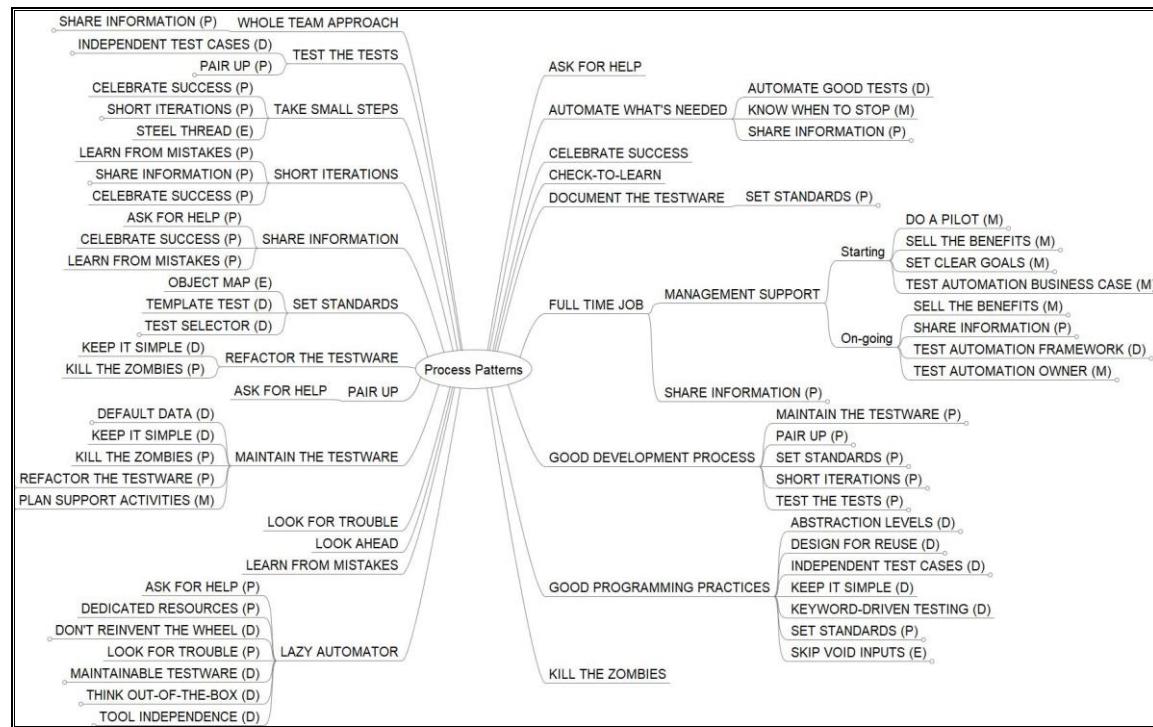


Figure 2.1.3-1- Process Patterns

Management Patterns

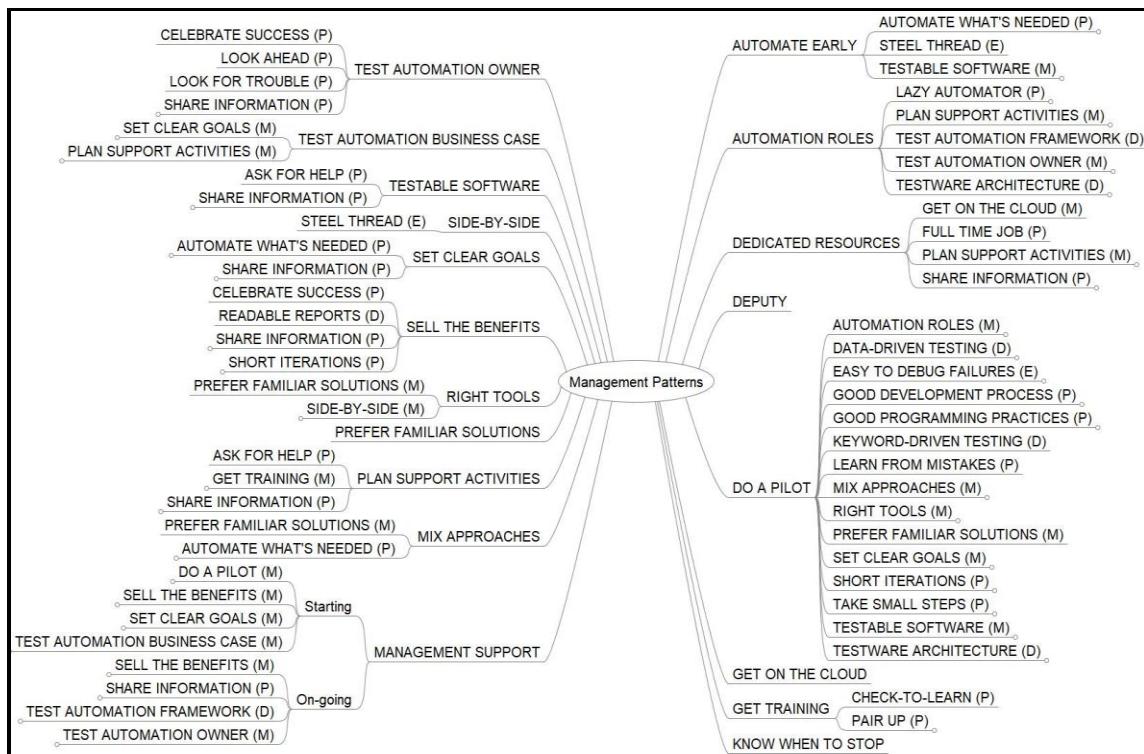


Figure 2.1.3-2 - Management Patterns

Design Patterns

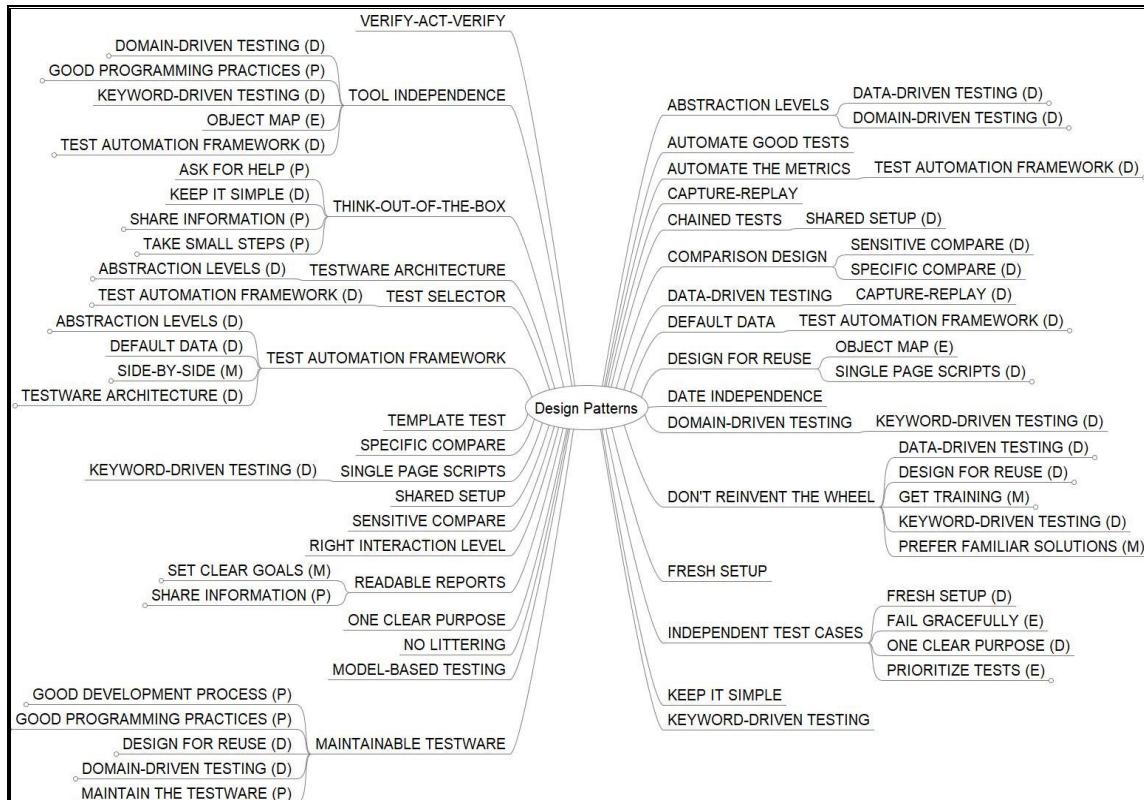


Figure 2.1.3-3 - Design Patterns

Execution Patterns

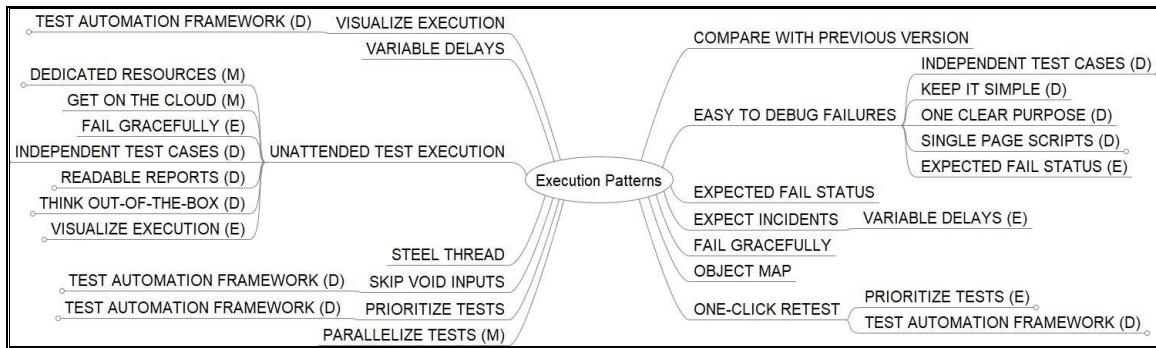


Figure 2.1.3-4 - Execution Patterns

3. Part 3

3.1. Issues in Detail

In this section, we show all of the Issues, listed in alphabetical order, so you can find a relevant issue easily if you know its name.

This section repeats what is in the wiki at the time of producing this book (2018). The wiki will have the most up-to-date versions of all issues (and patterns).

The wiki is much easier to use, as all other issues and patterns are links, but we wanted the book to be stand-alone, so it could be used without having access to the wiki (and the internet).

3.1.1. AD-HOC AUTOMATION (Management Issue)

Issue Summary

Automation is done ad-hoc with no planning or preparation.

"Ad hoc" means something created or done only for a particular purpose, impromptu, expedient, improvised, makeshift, "rough and ready".

Category

Management

Examples

1. Management thought that buying a tool was enough. Goals, architecture, resources etc. have not been planned or considered.
2. Expenses for test automation resources have not been budgeted.
3. There are not enough machines on which to run automation.
4. The databases for automation have to be shared with development or testing.
5. There are not enough tool licences.
6. Automation is done "on the side" when one has time to spare.
7. Support from testers, developers or other specialists has not been planned, so they have no time to help.

Questions

Who is doing automation? Is somebody in charge?

Does management know about it? Do they support it?

Resolving Patterns

Most recommended:

- TEST AUTOMATION OWNER: Appoint an owner for the test automation effort. If there is already a "champion" give him or her public support. The owner, once test automation has been established, controls that it stays

"healthy" and keeps an eye on new tools, techniques or processes in order to improve it.

- MANAGEMENT SUPPORT: you need this pattern in order to be able to apply the other patterns.
- DEDICATED RESOURCES: helps you get the resources you need.
- WHOLE TEAM APPROACH: if your developer team uses an agile development process, this is the pattern of choice.

Other useful patterns:

- SET CLEAR GOALS: Define the automation goals from the very beginning and when you want to re-vitalise a stalled automation effort.
- DO A PILOT: Start a pilot project to explore how to best automate tests for the application.
- TEST AUTOMATION BUSINESS CASE: this pattern may be needed when you start planning test automation or to justify making changes to existing automation that has stalled or not given sufficient benefit.

3.1.2. AUTOMATION DECAY (Process Issue)

Issue Summary

Automation hasn't been or is not properly maintained and gradually decays to shelfware.

Category

Process

Examples

Common causes of decay include (a combination of):

1. Business pressures: management pushes only to automate new test cases so that there is no time to maintain existing automation testware.
2. Lack of process or understanding: management doesn't understand the need for maintenance, and makes decisions without considering the implications.
3. Lack of documentation: automation testware is created without the necessary supporting documentation.
4. Lack of collaboration: knowledge isn't shared around the organization and business efficiency suffers, or junior automators are not properly mentored.
5. Delayed refactoring: adjustments in the automation testware due to changes in the Software under Test (SUT) are postponed. Delays mean that more effort is later required to bring the testware up-to-date.
6. Lack of knowledge: automators simply don't know how to write maintainable automation or the ones that knew have left the team or the company.

Questions

Who is in charge of automation?
Who sets the priorities?
Is there an automation team? How is it composed?

Resolving Patterns

Most recommended:

- TEST AUTOMATION OWNER: If there isn't yet a "champion" for test automation, it's high time to do so. Find who would do, and appoint him or her publicly.
- REFACTOR THE TESTWARE: If the current automation is becoming less useful, then it needs to be changed so that it meets the current needs and goals of the testing and the automation.

Other useful patterns:

- ASK FOR HELP: This pattern is a no-brainer! Use it.
- LOOK AHEAD: keep in touch with the tester, automation and development community in order to stay informed about new tools, methods etc.
- LOOK FOR TROUBLE: watch out for possible issues in order to solve them before they become unmanageable.
- MANAGEMENT SUPPORT: management support is a key resource when fighting against automation decay. Without it you will only be able to win a few "battles", but not the "war".

3.1.3. BRITTLE SCRIPTS (*Design Issue*)

Issue Summary

Automation scripts have to be reworked for any small change to the Software Under Test (SUT).

Category

Design

Examples

1. Scripts are created using the capture functionality of an automation tool. If in the meantime something has been changed in the application, the tests will break unless recorded anew.
2. A small change to the application (such as moving something to a different screen or changing the text of a button) causes many scripts to fail because this information is embedded in the scripts for many tests.

Questions

How do you develop automation scripts? (Capture is not good long-term though ok for short complex actions.)

Is there repeated code in any scripts? (Keep your automation code DRY - Don't Repeat Yourself - put common code and actions into scripts called by other scripts.) What kinds of changes are most likely to happen to the application? (Make your automation most flexible for those changes.)

Resolving Patterns

Most recommended:

- TESTWARE ARCHITECTURE: This pattern encompasses the overall approach to the automation artefacts and is best thought about right from the start of automation so that you can avoid having this issue. This pattern is implemented using:
 - ABSTRACTION LEVELS: This is the pattern to apply if you want to delegate some of the maintenance effort to the testers. It will enable you to write test cases that are both independent from the SUT and from the technical implementation of the automation (including the tools(s)).
- MAINTAINABLE TESTWARE: This is the pattern to apply if you want to get rid of the issue once and for all. If you haven't implemented it yet, you may want to apply at least some aspects of this pattern.
- MANAGEMENT SUPPORT: This is the pattern to apply if you are missing support or resources that you need in order to develop MAINTAINABLE TESTWARE.
- MODEL-BASED TESTING: This pattern involves considerable effort at the beginning but is the most efficient in the long run. Using a test model, the test cases can be cleanly separated from the technical details. Frequently used sequences of test steps can be defined as reusable and parametrisable building blocks. If the SUT changes, usually only a few building blocks need to be adapted while the test scripts are updated automatically.
- COMPARISON DESIGN: Design the comparison of test results to be as efficient as possible, balancing Dynamic and Post-Execution Comparison, and using a mixture of Sensitive and Robust/Specific comparisons.

Other useful patterns:

- GOOD PROGRAMMING PRACTICES: This pattern should already be in use! If not, you should apply it for all new automation efforts. Apply it also every time you have to change current testware.

3.1.4. BUGGY SCRIPTS (Process Issue)

Issue Summary

Scripts are not tested adequately.

Category

Process

Examples

1. Automation scripts are generated with a tool (for instance with a capture functionality).
2. Scripts are deemed so simple that there is no need to test them.
3. Scripts are written by an expert that knows what he is doing.

Questions

Who is writing the scripts? How?

Who is going to use them?

Resolving Patterns

Most recommended:

- TEST THE TESTS: Test the scripts just as you would test production code.
- GOOD PROGRAMMING PRACTICES: Use the same good programming practices as in software development.
- GOOD DEVELOPMENT PROCESS: Use good development practices for test code; the same as developers (should) use for production code.

3.1.5. CAN'T FIND WHAT I WANT (*Design Issue*)

Issue Summary

There is a script or file or dataset, but you don't remember what it's called or where to find it.

Category

Design

Examples

1. There is a script that initialises a data file that you want to use for a test. You have used it before but don't remember its name. You have looked through a long list of names but didn't recognise it.
2. There is a useful little utility that will do just what you want, but you have no idea where you put it.
3. You want to change the interface to a script or a keyword that you know is being used somewhere, and you need a usage list to make sure you update all the occurrences.

Questions

How often does this happen?

What happens when you can't find something - do you make another (duplicate) one?

Does this happen also in instances like example 3?

Resolving Patterns

Most recommended:

- It is important to set out the "rules" for standard ways of identifying and naming automation artefacts, ideally from the beginning. This is one of the things to investigate when you DO A PILOT.
- The TESTWARE ARCHITECTURE defines where different types of things are stored, so that you know where to go to find them.
- It is important to SET STANDARDS for things such as how the artefacts are documented, naming conventions and other things that everyone has to deal with. For example, any script should state its purpose, its prerequisites, and what it produces. This enables you to know whether or not it is actually suitable for you.
- For Example 3 you should check that your tool or framework is capable of compiling a usage list.

Other useful patterns:

- SHARE INFORMATION: Ask for and give information to managers, developers, other testers and customers.

3.1.6. COMPLEX ENVIRONMENT (Design Issue)

Issue summary

The environment where the Software Under Test (SUT) has to run is complex.

Category

Design

Examples

The problem range is:

1. Different systems (hardware or software) that interact in some way.
2. Test data is hard to create due to the number of systems involved or a limited refresh cycle.
3. You have to simulate some hardware.

Questions

Where does the complexity lie? In preparing the initial conditions? Test execution?

Checking the results? Dependencies?

How do you collect the necessary data?

Resolving Patterns

Most recommended:

- DO A PILOT: use this pattern to find out what the problems are and ways to tackle them.
- TAKE SMALL STEPS: use this pattern to break up the problems in more chewable chunks.
- THINK OUT-OF-THE-BOX: try to look at the problem from unusual viewpoints.

Other useful patterns:

- KEEP IT SIMPLE: this pattern is always helpful.
- SHARE INFORMATION: use this pattern for better communication between development, testing and automation

3.1.7. DATA CREEP (*Process Issue*)

Issue Summary

There are countless data files with different names but identical or almost identical content.

Category

Process

Examples

1. Nobody knows what is being used and where, so nobody wants to be responsible for deleting eventually needed data.
2. To edit or remove the data files is too much work: one has to look up all the places where they are used and change the referrals. If the files are similar rather than identical, a unified file has to be created.

Questions

Is the data documented?

Are there standards regarding naming and documentation?

Who creates the data? How? Who uses it?

Resolving Patterns

Most recommended:

- GOOD PROGRAMMING PRACTICES: Use the same good programming practices as in software development.
- MAINTAINABLE TESTWARE: Design your testware so that it does not have to be updated for every little change in the Software Under Test (SUT).
- MAINTAIN THE TESTWARE: Maintain test automation scripts and test data regularly and from the very beginning.
- MANAGEMENT SUPPORT: You will need this pattern to be able to change the current bad behaviour.

- REFACTOR THE TESTWARE: Refactor your testware regularly.

You should already be applying these patterns. If not, do it!

Other useful Patterns:

- GOOD DEVELOPMENT PROCESS: apply this pattern if you don't have a process for developing test automation. Apply it also if your process lives only on paper (nobody cares).
- LEARN FROM MISTAKES: apply this pattern to turn mistakes into useful experiences.
- KILL THE ZOMBIES: Apply this pattern for a start.
- DEFAULT DATA: use this pattern if your tests use a lot of common data that is not relevant to the specific test case.
- DOCUMENT THE TESTWARE: you should be already applying this pattern. Retrofixing documentation is quite an effort. Do it in the future for all new projects and everytime you have to update something old.
- KEEP IT SIMPLE: Always apply this pattern!

3.1.8. DATE DEPENDENCY (*Design Issue*)

Issue Summary

Tests are dependent on a specific date.

Category

Design

Examples

1. Test cases check age or time passed since something happened.
2. The expected results consist of prints that also show the creation date.

Questions

Can the test cases be rewritten to avoid the date problem?

Can the expected results be checked in some other way?

Resolving Patterns

Most recommended:

- DATE INDEPENDENCE: Apply this pattern in order to solve this specific issue.

Other useful patterns:

- KEEP IT SIMPLE: Always apply this pattern!

3.1.9. FALSE FAIL (*Execution Issue*)

Issue summary

The tests fail not because of errors in the SUT, but because of errors in the test automation testware or environment issues

Category

Execution

Examples

1. Tests fail because a window pops up that was not considered when developing the automation.
2. Tests fail because the initial conditions have been corrupted by another test.
3. A test always passes if it runs by itself but always fails when run in the test suite.
4. Tests fail because something else is running on the same machine.
5. Tests fail because the database has been corrupted or changed by another application.
6. Tests fail because you are using a SENSITIVE COMPARE and so changes that have nothing to do with your test case affect the results.

Questions

Are the initial conditions set correctly?

Are there dependencies between the test cases?

Have the tests been sufficiently tested?

Resolving Patterns

Most recommended:

- COMPARISON DESIGN: design the comparison of test results to be as efficient as possible, balancing Dynamic and Post-Execution Comparison, and using a mixture of Sensitive and Robust/ Specific comparisons.
- DEDICATED RESOURCES: Use this pattern if you have issues similar to Examples 4. or 5.
- FRESH SETUP: This pattern guards against issues like Example 2.
- INDEPENDENT TEST CASES: apply this pattern to get rid of issues like Example 3.
- RIGHT INTERACTION LEVEL: make sure that your tests interact with the SUT at the most effective level.

Other useful patterns:

- MAINTAIN THE TESTWARE: apply this pattern to make sure that your automation keeps working even if the SUT is being changed.
- SHARE INFORMATION: this pattern helps you learn in time when development is planning big or small changes that affect automation.

- SPECIFIC COMPARE: Expected results are specific to the test case so changes to objects not processed in the test case don't affect the test results. Use this pattern for issues like Example 6.
- TEST THE TESTS: use this pattern always (even for quick fixes!).

3.1.10. FALSE PASS (*Execution Issue*)

Issue summary

The tests pass even if the SUT actually reacts erroneously.

Category

Execution

Examples

1. The automated tests drive the SUT, but don't compare the results with expected results so that they always pass.
2. The new results are erroneously written over the expected results so that the compares always pass.
3. Test automation has been developed with an erroneous SUT and the results have been accepted as expected results without first checking that they are correct.
4. The tests aren't picking up bugs even though they are comparing the results for the right things and should find them.

Questions

How are the expected results collected?

Have the tests been sufficiently tested?

Resolving Patterns

Most recommended:

- COMPARISON DESIGN: design the comparison of test results to be as efficient as possible, balancing Dynamic and Post-Execution Comparison, and using a mixture of Sensitive and Robust/ Specific comparisons.
- RIGHT INTERACTION LEVEL: make sure that your tests interact with the SUT at the most effective level.
- TEST THE TESTS: This is an important pattern to make sure that your tests are doing what you think they are doing.
- VERIFY-ACT-VERIFY: use this pattern if you cannot apply FRESH SETUP.

Other useful Patterns:

- MAINTAIN THE TESTWARE: use this pattern to keep your automation workable.

- SENSITIVE COMPARE: If the tests are missing bugs because they are comparing too narrow a set of results (e.g. only the changed fields), the use of sensitive comparison can find unexpected results, where the test (correctly) fails.

3.1.11. FLAKY TESTS (*Execution Issue*)

Issue summary

The automated tests seem to pass or fail randomly.

Category

Execution

Examples

1. Mostly the tests pass but once in a while they fail. If they are rerun, they pass as usual.
2. Automated tests fail because development was using your same database and changed its structure.
3. Testers changed the files test automation uses to compare results and now those tests don't pass any longer.
4. IT-Support shut down your machine to install some new software and killed your automation run.
5. IT-Support installed new software and now your tool doesn't work any more.
6. Automated tests run in parallel using the same databases and mess up each other's data.
7. A previous test failed and left the system "dirty" so that all successive tests also failed.
8. Automated tests assume that something in the test environment is immutable, i.e. will not change, because the setup is assumed to be under the testers' control and deviations can be corrected when spotted.
(contributed by Derek Bergin).

Questions

Do the tests fail whenever a preceding test was also run? Or whenever a preceding test was not run?

What runs at the same time as the automated tests?

Did IT-Support run some updates just as the automated tests were running?

Was somebody working on the same database as the automated tests just as they were running?

Have the automated tests been sufficiently tested?

Who uses which machines and when? Is it possible to plan when each group has access to the machines?

Is there a central database for everybody or does each user or group have its own?

What are you depending on not changing in the test environment (what do you assume is immutable)? How will you monitor changes in the test environment (expected and unexpected)? Can you investigate the environment after a failure?

Resolving Patterns

The following patterns can all be recommended:

- DEDICATED RESOURCES: this is the pattern of choice for this issue.
- FAIL GRACEFULLY: use this pattern if you suspect your issue to be like Example 7.
- INDEPENDENT TEST CASES: use this pattern if your test automation suite is tripping itself up.
- RIGHT INTERACTION LEVEL: make sure that your tests interact with the SUT at the most effective level.
- TEST THE TESTS: this is really a no-brainer. Use it always.
- VERIFY-ACT-VERIFY: use this pattern if you cannot apply FRESH SETUP.
- If different execution times are the reason for the 'flakiness' use the pattern VARIABLE DELAYS.

3.1.12. GIANT SCRIPTS (Design Issue)

Issue Summary

Scripts that span thousands of lines.

Category

Design

Examples

1. Scripts are not modular; one script does the test case from beginning to end (this is often the case if you use CAPTURE-REPLAY).
2. Scripts perform more than one test case.

Questions

Who is doing the scripting?

Resolving Patterns

Most recommended:

- ABSTRACTION LEVELS: This pattern helps but it may be too costly or too time-consuming to change retroactively the architecture of your current automation, still you should use this pattern for all new automation efforts or when you have to do complex updates.
- GOOD DEVELOPMENT PROCESS: Apply this pattern if you don't have a process for developing test automation. Apply it also if your process lives only on paper (nobody cares).

- GOOD PROGRAMMING PRACTICES: This pattern should already be in use! If not, you should apply it for all new automation efforts. Apply it also every time you have to change current testware.
- ONE CLEAR PURPOSE: Use this pattern if you have scripts that perform more than one test case.

Other useful patterns:

- DESIGN FOR REUSE: Apply this pattern for long lasting testware.
- KEEP IT SIMPLE: Use this pattern always.
- SINGLE PAGE SCRIPTS: Use this pattern to modularize your scripts.

3.1.13. HARD-TO-AUTOMATE (*Design Issue*)

Issue summary

The Software Under Test (SUT) doesn't support automated testing very well or not at all.

Category

Design

Examples

The problem range is:

1. Non-identifiable (uniquely) GUI-Objects; caused by lack of unique name or other property to make it unique.
2. Third Party or Customized GUI-Objects that are not recognized properly or are not 'open' (exposed methods and properties).
3. Random processing.
4. Fluctuating response times.
5. The expected results change after each new version of the SUT.
6. Different systems (hardware or software) that interact in some way.
7. Tests have to run on a steadily growing number of browsers or environments.
8. Test results consist in entries in many database tables or files related to each other.
9. For embedded systems or mobile devices results are difficult to check because of timing issues or not very visible results (e.g. intermediate results).
10. Test data is hard to create due to the number of systems involved or a limited refresh cycle.
11. Unexpected pop-ups.
12. The SUT is very slow so it takes a lot of time to test the automation scripts.
13. APIs or AI deliver non-predictable results.

Questions

Where does the complexity lie? In preparing the initial conditions? Test execution?
Checking the results? Dependencies?
Do you have the right resources? What do you need?
Can you simulate hardware? or software?
Has the current tool been selected especially for the SUT or has it been "inherited"
from previous applications?
Do the developers know about the automation problems? Do they care?
Do the developers have guidelines/standards for building in 'testability'? If not, why
not?
How do manual testers check the results?
How complex are the test cases?
How do you collect the necessary data?

Resolving Patterns

Most recommended:

- DO A PILOT: use this pattern to find out what the problems are and ways to tackle them.
- RIGHT TOOLS: use this pattern if you are confronting issues like Examples 1 or 2.
- TAKE SMALL STEPS: use this pattern to break up the problems in more chewable chunks.
- TESTABLE SOFTWARE: use this pattern for issues similar to Examples 1, 2. and 3.
- THINK OUT-OF-THE-BOX: try to look at the problem from unusual viewpoints. This is especially important for issues like Example 13.
- VARIABLE DELAYS: use this pattern for issues Like Example 4.

Other useful patterns:

- EASY TO DEBUG FAILURES: use this pattern for issues like Example 8.
- GET ON THE CLOUD: this is the pattern of choice for issues like Example 7.
- KEEP IT SIMPLE: this pattern is always helpful.
- SHARE INFORMATION: use this pattern for better communication between development, testing and automation (Examples 1, 2, 4., 5., 11. and 12).

3.1.14. HARD-TO-AUTOMATE RESULTS (Design Issue)

Issue Summary

Preparing the expected results is slow and difficult.

Category

Design

Examples

1. Results can only be compared on the GUI.
2. Results are spread through different databases or media and it's difficult to put them together in a meaningful way.
3. Results change randomly.
4. Results change from release to release.
5. Results depend on what test cases have been run before.
6. API or AI tests deliver non-predictable results.

Questions

How complex are the automated test cases? Can they be split up?
Are complex "scenario" test cases really necessary?

Resolving Patterns

Most recommended:

- FRESH SETUP: apply this pattern if you have issues like Example 5.
- DEDICATED RESOURCES: look up this pattern if you have to share resources.
- THINK OUT-OF-THE-BOX: try to look at the problem from unusual viewpoints. This is especially important for issues like Example 6.
- WHOLE TEAM APPROACH: if your team follows an agile development process this is the pattern to use in order to avoid this kind of problems from the beginning.

Other useful patterns:

- SHARE INFORMATION: use this pattern for better communication between development, testing and automation.
- DESIGN FOR REUSE: this pattern should be used always.
- COMPARE WITH PREVIOUS VERSION: this pattern helps when nothing else does.
- TAKE SMALL STEPS: this pattern is always useful.

3.1.15. HIGH ROI EXPECTATIONS (*Management Issue*)

Issue Summary

Management wants to maximize ROI from the test automation project, but is not prepared to invest adequately.

Category

Management

Examples

1. Management thinks that buying an expensive testing tool is all that's needed for test automation.

2. Management expects that with test automation you will be able to find lots of bugs (even if it is regression testing that was automated).
3. Management thinks that having test automation means that fewer testers will be needed.
4. Management thinks that you will never have to change tools, so why put so much effort into the architecture of the testware.

Questions

What does management expect from test automation? What do testers expect? Has a business plan been worked out?

Resolving Patterns

Most recommended:

- AUTOMATE GOOD TESTS: use this pattern to select the test cases to automate that will bring the greatest ROI as quickly as possible.
- Make sure that you have SET CLEAR GOALS which are realistic and achievable.
- DO A PILOT: apply this pattern if you are just starting with test automation and neither management nor you really can judge what it costs and what it will bring, or if previous automation has been too expensive and you want to make significant changes to improve the ROI.
- MAINTAINABLE TESTWARE: apply this pattern for long-term ROI, not necessarily for quick wins.
- SHARE INFORMATION: communicate what can realistically be achieved with automation, set manager's expectations to more realistic levels.

Other useful patterns:

- AUTOMATE EARLY: This pattern, together with WHOLE TEAM APPROACH, can help you deliver solid automation from the project start.
- DESIGN FOR REUSE: Apply this pattern for long-term ROI.
- GOOD PROGRAMMING PRACTICES: Apply this pattern for long-term ROI.
- MANAGEMENT SUPPORT: You will need this pattern in order to be able to implement other patterns (for instance WHOLE TEAM APPROACH or DO A PILOT).
- TAKE SMALL STEPS: Use this pattern for quick wins, but actually it's always applicable.
- TEST AUTOMATION FRAMEWORK: Good pattern for long-term ROI.
- TOOL INDEPENDENCE: Apply this pattern if you want to implement really long-lasting automation.
- WHOLE TEAM APPROACH: If you are just starting with automation and your developers are using an agile process, then this is definitely the pattern of choice. Otherwise you must first convince management and development to switch to agile.

3.1.16. INADEQUATE COMMUNICATION (Process Issue)

Issue Summary

This issue covers two frequently recurring problems:

- Testers don't know what automation could deliver and the test automation team doesn't know what testers need.
- Developers don't understand, don't know or don't care about the effect of their changes on the automation.

Category

Process

Examples

1. Test cases that should be automated are written very sparingly because "everybody knows what you have to do"... only automators do not.
2. Automators need help from some tester or specialist, but that person is not available or doesn't have time.
3. Testers do a lot of preparations to do manual testing that could be easily automated if only the automators knew about it.
4. Testers, developers and automators work in different buildings, cities, time zones, or countries.
5. Developers change the Software Under Test (SUT) without caring if it disrupts the automation or makes it harder.

Questions

Are testers and automators on the same team? If not, why not?

Do developers notify automators when they want to use new components? Do automators report to development which components they cannot drive?

How often do team members meet personally? How often in telephone conferences / live meetings?

Do team members know each other? How about time or language differences?

Do team members with the same role have the same experience / know-how? Do they speak the same "language"?

Resolving Patterns

Most recommended:

- SHARE INFORMATION: this pattern is a no brainer for big and small automation efforts. Use it!
- WHOLE TEAM APPROACH: if your development team uses an agile process and you apply this pattern, you will not encounter this issue.

Other useful patterns:

- GET ON THE CLOUD: This pattern is especially useful if you are working with a distributed team.

3.1.17. INADEQUATE DOCUMENTATION (*Process Issue*)

Issue Summary

Test automation testware (scripts data etc.) are not adequately documented.

Category

Process

Examples

1. Test scripts are not documented at all or not adequately, so they cannot be found (no reuse) or nobody knows what they actually do.
2. Available data is not documented so it cannot be reused.
3. Documentation is missing about the kind of data and / or where to find it or where to store it.
4. There is no overview of the automation "regime" - where various artefacts are stored, the aims and objectives of the automation, etc.

Questions

Who creates the scripts? Who maintains them?

Who creates the data? Who maintains it?

Who uses the data or the scripts?

Are there standards? Are they used?

Resolving Patterns

Most recommended:

- DOCUMENT THE TESTWARE: Document the automation scripts and the test data.
- DESIGN FOR REUSE: Design reusable testware.

Other useful patterns:

- SET STANDARDS: Set and follow standards for the automation artefacts.

3.1.18. INADEQUATE RESOURCES (*Execution Issue*)

Issue Summary

The hardware or the network are too slow.

Category

Execution

Examples

1. Automated tests run on older machines that have been discarded by developers.
2. The network is overloaded so response times are way too long.

3. Databases are shared with testing and development and on critical times can slow down to a creep.

Questions

Why aren't the appropriate resources available?

Resolving Patterns

Most recommended:

- DEDICATED RESOURCES: Use this pattern in order to get the resources that you need, when you need them.
- MANAGEMENT SUPPORT: Support from management is needed in order to get the resources needed for optimal test execution.

Other useful patterns:

- GET ON THE CLOUD: Getting on the cloud can be a very economical way to solve bottlenecks.

3.1.19. INADEQUATE REVISION CONTROL (*Process Issue*)

Issue Summary

Test automation scripts are not consistently paired to the correct version of the Software Under Test (SUT).

Category

Process

Examples

1. There is no revision control for the automated scripts, so that it's not possible to run tests for different versions of the SUT.
2. There is no revision control, not even for the SUT.

Questions

Does development have revision control?

Who manages revision control? Development?

Resolving Patterns

Most recommended:

- GOOD DEVELOPMENT PROCESS: Use good development practices for test code; the same as developers (should) use for production code.
- SHARE INFORMATION: Ask for and give information to managers, developers, other testers and customers.

3.1.20. INADEQUATE SUPPORT (Management Issue)

Issue summary

The test automation team doesn't get adequate support from management, testers or other specialists.

Category

Management

Examples

1. Management knows too little about the benefits of test automation or about the investment needed in time or money to achieve good automation.
2. Testers don't have time to write test cases to be automated or to otherwise support the test automation team.
3. The test automation team never gets time for maintenance.
4. You need support (from IT-specialists, developers, database experts, business analysts etc), but nobody has time for you.
5. Expenses or time for training haven't been budgeted.

Resolving Patterns

Most recommended:

- Automation started by one person does not often "scale up". You may need to use this experience as a learning exercise and start again with good MANAGEMENT SUPPORT. This pattern also applies if the automation work does not have a high enough priority, compared to other tasks.
- When support from other people is not there, this could also be because of a failure to PLAN SUPPORT ACTIVITIES. If you end up starting again, be sure to use this pattern, which may be more difficult if your team is distributed over different geographical locations.

Other useful patterns:

- SELL THE BENEFITS: use this pattern again and again to show everyone what the automation effort can achieve with their support.
- SHARE INFORMATION: this pattern may help explain why people have no time for you. It is important to let people know why and when you need their support.
- SHORT ITERATIONS: use this pattern to work with short feedback cycles. This will let people who are supporting the automation effort know more quickly that their support is enabling you to achieve results, and this will encourage further support.

3.1.21. INADEQUATE TEAM (Management Issue)

Issue Summary

Either there is no test automation team and it must be built from scratch or you do not have all the roles you need on board or some or all the members of the test automation team are not suited to do test automation.

Category

Management

Examples

1. Test automation is currently done by a "lone cowboy", but management wants to broaden the scope by creating a test automation team.
2. Test automation doesn't proceed because nobody really has command of the current tool or of some other technical issue.
3. The person who had started some automation has now left the company, and no one wants to take it up (often associated with a tool becoming "shelf-ware").
4. A tester that has no aptitude for development is supposed to develop automated scripts.
5. A developer that doesn't want to, is forced by management to do automation.
6. There is a team, but nobody is in charge, so people do just what they like.
7. Scripts are written by people with minimal domain knowledge, resulting in tests that are not reflecting reality. (Patrick de Beer).

Questions

How is the automation team selected? By whom?

Do you have a team yet? Who is on it?

What roles do you have? Which are missing?

Resolving Patterns

Most recommended:

- AUTOMATION ROLES: this pattern helps to select the right people with the right skills for automation.
- SET CLEAR GOALS: Apply this pattern if you are just starting with test automation.
- WHOLE TEAM APPROACH: if the development team uses an agile process, then this pattern will help even unsatisfied people to better integrate.

Other useful patterns:

- FULL TIME JOB: this pattern helps the automation team work more efficiently.

3.1.22. INADEQUATE TECHNICAL RESOURCES (Management Issue)

Issue Summary

The technical resources for test automation are insufficient or not appropriate or both.

Category

Management

Examples

1. Sharing machines, databases or network with development or testing causes your automated tests to fail unpredictably (FLAKY TESTS).
2. The network is so slow that your automated tests take too long to run.
3. You cannot run your tests on the same environments as your customers.

Questions

Who is in charge of the technical side of test automation?

Resolving Patterns

Most recommended:

- DEDICATED RESOURCES for example 1.
- GET ON THE CLOUD for examples 2 and 3.

3.1.23. INADEQUATE TOOLS (Management Issue)

Issue summary

The tools that are currently used are not appropriate for the Software Under Test (SUT) or are not supported any longer or you still don't have adequate tools.

Category

Management

Examples

1. The test automation tool that you are currently using is not supported any longer.
2. Development has implemented in the SUT components that are not supported (well) by your current tool.
3. The only person that could use the current tool left the company.
4. You want to start with test automation and have to select one or more suitable tools.

Questions

Since how long are you using your current tool?

Who is using the tool?

What automation architecture are you using?

Resolving Patterns

Most recommended:

- **MIX APPROACHES:** This pattern helps you recognize what is best done with a tool and what not.
- **RIGHT TOOLS:** Apply this pattern when you select a tool to start with test automation or when you want to change the currently used tool.
- **TEST AUTOMATION FRAMEWORK:** consider this pattern when you need to coordinate the different tools that you are currently using.

Other useful patterns:

- **LEARN FROM MISTAKES:** this pattern helps you turn bad experiences into learning experiences.
- **MANAGEMENT SUPPORT:** this pattern will help you get the necessary support to get the **RIGHT TOOLS**.
- **PREFER FAMILIAR SOLUTIONS:** use this pattern if a possible tool is already used / known in your company.
- **SELL THE BENEFITS:** use this pattern, when you have selected a tool to show its advantages over other ones.
- **SIDE-BY-SIDE:** This pattern helps when having to pick a tool from a larger selection
- **TESTABLE SOFTWARE:** apply this pattern when starting with test automation.

3.1.24. INCONSISTENT DATA (*Design Issue*)

Issue summary

The data needed for the automated test cases changes unpredictably.

Category

Design

Examples

1. Test data is obtained by anonymising customer data. This has some major disadvantages:
 - a. The data must be anonymised (this can be tricky).
 - b. You must be able to use different IDs or search criteria for every run.
 - c. You probably won't get just the data combination that you need for a special test case and if you add it, it will be overwritten next time you get a new batch of customer data.

- d. Different input data means different output data, so it will be more difficult to compare results.
- 2. Test automation shares databases with testing or development.
- 3. After an update to the Software Under Test (SUT) the data is not compatible any longer.
- 4. Tests alter the data for following tests.

Questions

How do you collect the necessary data?

Resolving Patterns

Most recommended:

- FRESH SETUP should help you in most cases.
- DEDICATED RESOURCES: look up this pattern if you have to share resources.
- WHOLE TEAM APPROACH: if your team follows an agile development process this is the pattern to use in order to avoid this kind of problems from the beginning.

Other useful patterns:

- SHARE INFORMATION: use this pattern for better communication between development, testing and automation.
- DESIGN FOR REUSE: this pattern should be used always.

3.1.25. INEFFICIENT EXECUTION (Execution Issue)

Issue summary

Execution of automated tests is too slow or too cumbersome.

Category

Execution

Examples

- 1. Automated tests run slowly because synchronisation with the user interface involves waiting for a response.
- 2. The automated tests take so long to run that they are executed only for the final release, not in a daily build.
- 3. Test set-up and post-processing hasn't been automated, requires some manual intervention or is done in an inefficient way.
- 4. Execution is disrupted by unexpected windows (or pop-ups).

Questions

How long do the tests take to run? Why is this a problem? (E.g. is there a time window that a set of tests should be run in but they take longer than that?)

How much of the test execution time is spent on activities other than running tests, e.g. set-up, clear-up?

Can tests be run in parallel? Can set-up etc be separated out to separate scripts?

Are too many tests run through the Graphical User Interface (GUI) rather than through an Application Programming Interface (API)?

Resolving Patterns

Most recommended:

- PARALLELIZE TESTS: split the tests to run in parallel (on different machines), which requires INDEPENDENT TEST CASES.
- STEEL THREAD: Use this pattern to help define a minimal set of tests to be run.
- TEST SELECTOR: Implement your test cases so that you can turn on various selection criteria for whether or not a given test is included in an execution run.
- EXPECT INCIDENTS: automated scripts should be able to react to unexpected incidents without disrupting execution.
- VARIABLE DELAYS: Use this pattern if execution times depend for instance on the state of the nework or on the different lenght of some SUT-processes.

Other useful patterns:

- UNATTENDED TEST EXECUTION: This pattern is what you want to automate your tests for! Use it!
- SKIP VOID INPUTS: in order to make your scripts more efficient, arrange for an easy way to automatically skip void Inputs.

3.1.26. INEFFICIENT FAILURE ANALYSIS (*Execution Issue*)

Issue summary

Failure Analysis is slow and difficult.

Category

Execution

Examples

1. Automated test cases perform complex scenarios, so that it's difficult to tag the exact cause for a failure.
2. Automated test cases depend on previous tests. If the failure is caused in a previous test it is very difficult to find it.

3. Failures are caused by differences in long amorphous text files that are difficult to parse manually (even with compare tools).
4. In order to examine the status of the SUT after a test has been run, it's often necessary to rerun it. If tests routinely clean up after themselves, you have to remove temporarily the clean up actions before rerunning the test.
5. Results can only be compared on the GUI.
6. Results are spread through different databases or media and it's difficult to put them together in a meaningful way.
7. Results change randomly.
8. Results change from release to release.
9. A minor bug which isn't going to be fixed is making too many of the automated tests fail, but they still need to be looked at (and then ignored).
10. Comparison of results takes a long time, or results which should abort a test run are not picked up until the test has finished.

Questions

How complex are the automated test cases? Can they be split up?

Are the failures in the SUT or in the automation?

Are complex "scenario" test cases really necessary?

What information do developers need to facilitate bug-fixing?

What information do automators need to isolate why an automated test has failed?

What information do testers need to determine the cause of a test failure?

Resolving Patterns

Most recommended:

- EASY TO DEBUG FAILURES: This is the pattern you need to solve this issue.
- COMPARISON DESIGN: Are you using the right type and sensitivity of result comparisons? Use this pattern to make sure your test result comparisons are as efficient as possible.
- KEEP IT SIMPLE: Apply this pattern always.
- EXPECTED FAIL STATUS: Use this pattern when a minor bug is causing many automated tests to fail (Example 9).
- ONE-CLICK RETEST: Apply this pattern to solve issues like Example 3.
- READABLE REPORTS: This pattern together with EASY TO DEBUG FAILURES should help to solve this issue.

Other useful patterns:

- COMPARE WITH PREVIOUS VERSION: this pattern makes it easy to find failures, but not to analyse them.
- FRESH SETUP: apply this pattern if you have issues like Example 2.
- TAKE SMALL STEPS: this pattern is always useful.

3.1.27. INADEQUATE TEAM (*Management Issue*)

Issue Summary

Either there is no test automation team and it must be built from scratch or you do not have all the roles you need on board or some or all the members of the test automation team are not suited to do test automation

Category

Management

Examples

1. Test automation is currently done by a "lone cowboy", but management wants to broaden the scope by creating a test automation team.
2. Test automation doesn't proceed because nobody really has command of the current tool or of some other technical issue.
3. The person who had started some automation has now left the company, and no one wants to take it up (often associated with a tool becoming "shelf-ware").
4. A tester that has no aptitude for development is supposed to develop automated scripts.
5. A developer that doesn't want to, is forced by management to do automation.
6. There is a team, but nobody is in charge, so people do just what they like.
7. Scripts are written by people with minimal domain knowledge, resulting in tests that are not reflecting reality. [Patrick de Beer].

Questions

How is the automation team selected? By whom?

Do you have a team yet? Who is on it?

What roles do you have? Which are missing?

Resolving Patterns

Most recommended:

- AUTOMATION ROLES: this pattern helps to select the right people with the right skills for automation.
- SET CLEAR GOALS: Apply this pattern if you are just starting with test automation.
- WHOLE TEAM APPROACH: if the development team uses an agile process, then this pattern will help even unsatisfied people to better integrate.

Other useful patterns:

- FULL TIME JOB: this pattern helps the automation team work more efficiently.

3.1.28. INFLEXIBLE AUTOMATION (*Execution Issue*)

Issue Summary

Execution of automated tests is ad-hoc or too rigid. Subsets of tests cannot be selected to run - it's "all or nothing".

Category

Execution

Examples

1. It is difficult to select a subset of tests in order to test only some special feature, or to run a different set of regression tests than last time.
2. Automated tests can be run only by one person. If he / she is on sick leave the tests don't run!
3. Your test suites cannot be used for smoke tests because all the tests build on each other.
4. To run an automated test again you have to make a lot of manual changes to the tests.

Questions

Do you know what every test does? Is this documented?

Who can run the tests? Who decides what tests to run?

How do the different test cases in a test suite relate to each other? Are there dependencies?

Do the tests have different priorities?

Resolving Patterns

Most recommended:

- TEST SELECTOR: Implement your test cases so that you can turn on various selection criteria for whether or not a given test is included in an execution run (example 1).
- UNATTENDED TEST EXECUTION: This pattern is what you want to automate your tests for! Use it!
- INDEPENDENT TEST CASES: Apply this pattern if your tests build on each other too much (example 3).
- TESTWARE ARCHITECTURE: Apply this pattern to avoid problems such as *MANUAL INTERVENTIONS* in automated testing (example 4).
- PRIORITIZE TESTS: Apply this pattern to take advantage of flexible execution, e.g. you have time to run some tests but not all of them.

Other useful patterns:

- DOCUMENT THE TESTWARE: use this pattern so that you know what each test does.

3.1.29. INTERDEPENDENT TEST CASES (*Design Issue*)

Issue Summary

Test cases depend on each other, that is they can only be executed in a fixed sequence.

Category

Design

Examples

1. Test cases must be executed in a fixed sequence because the preceding test cases create the initial conditions for the following ones.

Questions

Who designed the test cases, and for what purpose? Are manual test cases being automated "as is"?

If so, this can cause problems, because what makes sense for manual testing may not be the most appropriate solution for automated tests, particularly the dependencies of tests on each other. If one of a sequence of manual tests fails, the tester can usually find a way to continue testing, possibly by inserting the data that should have been produced by the failing test. But the tool will simply be "stopped in its tracks" by a failing test - it's not possible or practical to anticipate all possible ways that all tests can fail so that an automated solution can be implemented.

See also *MANUAL MIMICRY* - an issue describing what happens when you try to automate manual tests too literally.

Resolving Patterns

Most recommended:

- FRESH SETUP: use this pattern to initialize each test case independently.
- INDEPENDENT TEST CASES: this pattern is the best solution for this issue.

3.1.30. INSUFFICIENT METRICS (*Process Issue*)

Issue Summary

Few or no metrics are collected or not consistently. Collection is cumbersome.

Category

Process

Examples

1. Metrics are collected from different systems and have to be consolidated manually.
2. The collected metrics are not the metrics that are needed by management.

3. Metrics can only be collected after they have lost most of their importance.

Questions

What metrics are collected?

Which metrics does management need?

Who collects the metrics? When?

In what form are they saved?

Can the collection and/or presentation of the metrics be automated?

Resolving Patterns

Most recommended:

- **SHARE INFORMATION:** This pattern is important if you do not really know what metrics are needed (management, development, maintenance etc.).
- **AUTOMATE THE METRICS:** Apply this pattern if you need consistent and reliable metrics. It is most necessary for large testing efforts and not only for automation. For small projects it could become an overkill.

Other useful patterns:

- **SELL THE BENEFITS:** This pattern will help you get support to AUTOMATE THE METRICS.

3.1.31. KNOW-HOW LEAKAGE (*Management Issue*)

Issue summary

Test automation know-how (for instance scripting, tools) is being lost from the organisation.

Category

Management

Examples

1. Test automation was started by a "lone cowboy" who has left the company.
2. High turnover of staff means that people never get into sufficient detailed knowledge about the existing automation before they move on (and no backup was planned).
3. Test automation is shelfware and nobody knows what has been developed and where it has been stored.

Questions

Why are people leaving the team, the company?

Could someone who used to have good knowledge of the automation be called back to transfer knowledge to the current team? (as a consultant?)

Does it take too long to figure out the structure of the automation - is the architecture unclear?

Resolving Patterns

Most recommended:

- **DEPUTY:** Appoint a deputy to all important knowledge carriers.

Other useful patterns:

- **DOCUMENT THE TESTWARE:** This pattern is essential for keeping the test automation effort maintainable.
- **GET TRAINING:** this pattern should be used regularly by the whole team.
- **PAIR UP:** apply this pattern to train newbies as fast as possible.
- **SHARE INFORMATION:** apply this pattern to solve this issue effectively.

3.1.32. LATE TEST CASE DESIGN (Process Issue)

Issue Summary

Automated test cases are designed and written only after the Software Under Test (SUT) has been implemented.

There is a prevailing misconception that automation cannot start until the software being tested is "stable". Although there are some things that may need to wait until you know exactly what the interface is, far more can be done much earlier than many people realise.

Category

Process

Examples

1. The test automation efforts start only when the SUT has become quite stable, that is long after it has been developed.
2. Test automation starts only when the GUI has been developed, so no test cases are written in advance.

Questions

Who writes the test cases?

Who does test automation?

What things will need to be tested?

Are the proposed tests too close to the software and to the tool? Has any thought been given to layers of abstraction between the tests and the implementation of the tests?

Resolving Patterns

Most recommended:

- DOMAIN-DRIVEN TESTING: This pattern works quite well when testers and automators belong to different teams and have different skills.
- WHOLE TEAM APPROACH: this pattern takes care of issues like Examples 1. and 2., but it means that you have to completely overhaul how software is developed in your company. If you don't manage to get support for it, apply instead DOMAIN-DRIVEN TESTING.

Other useful patterns:

- KEYWORD-DRIVEN TESTING: use this pattern when you have issues like Example 1. and developers to code the necessary keywords.

3.1.33. *LIMITED EXPERIENCE (Management Issue)*

Issue summary

The test automation team has limited experience in test automation or in the SUT or new team members take too long to become productive

Category

Management

Examples

1. Test automation is planned, but nobody in the company has done it yet.
2. Only one person on the team knows the test tool.
3. Test automators know the tool, but not the SUT.
4. A new tool has been acquired.
5. New team members find it difficult to learn how to use the test tool.
6. Current team members don't have time to coach newcomers.
7. Testers don't know how to write test cases for automation.

Questions

Who is doing the automation? Testers? Developers?

Has the tool been selected by the automation team? Or is it shelfware and should be resuscitated?

Does the tool vendor offer trainings? Are the help manuals adequate? Are the help manuals in the current language?

Is the SUT adequately documented?

Is there a budget for training?

Resolving Patterns

Depending on your previous experiences you should first look up the patterns suggested here:

- Tester
The most important thing to learn about test automation is the difference between manual and automated tests.

Manual tests can:

- depend on each other if that makes it simpler for the tester (for instance one test creates the data for the next test)
- fail, but the tester, knowing what he or she has done, can easily describe the results. Also, the tester can go on with testing the next test case even if the first test left the system in an invalid state
- take different times to perform specific actions, but a tester knows when to wait and when to go on
- be performed on specific dates (testers can set and reset the date if needed)

Automated tests should:

- be written as INDEPENDENT TEST CASES so that they don't have to be executed together every time. See also PRIORITIZE TESTS, FRESH SETUP and TEST SELECTOR.
 - FAIL GRACEFULLY so that the next test cases can execute normally
 - have ONE CLEAR PURPOSE to make it easier to check the results. See also READABLE REPORTS, COMPARISON DESIGN, EXPECTED FAIL STATUS
 - reuse data whenever possible (see DEFAULT DATA, TESTWARE ARCHITECTURE)
 - use VARIABLE DELAYS to wait until some action in the System under Test (SUT) is finished instead of immediately moving on and therefore losing synchronisation
 - be able to perform date sensitive test cases (look up DATE INDEPENDENCE)
 - be written in such a way that changing the testing tool doesn't mean starting again from scratch. The patterns to look up in this case are TOOL INDEPENDENCE and OBJECT MAP
 - be maintainable. For instance, writing modular scripts. This means that actions that are performed in many test cases are written each in a specific script that can be called when needed. In this way changes affect only one script and not the whole testware (see GOOD PROGRAMMING PRACTICES, KEYWORD-DRIVEN TESTING, SINGLE PAGE SCRIPTS). Other important patterns are MAINTAINABLE TESTWARE, DESIGN FOR REUSE and ABSTRACTION LEVELS
 - Developer
The most important thing to learn about test automation is the difference between writing code to develop an application and code to automate tests
- Both systems (should) use:
- GOOD PROGRAMMING PRACTICES
 - GOOD DEVELOPMENT PROCESS

When developing an application, one should:

- write the application with the end user in mind
- give the user the possibility to select the next action
- show with infos, warnings or error messages how the application is doing
- document the code (best using unit tests)
- do pair programming
- refactor the code.

When developing test automation one should:

- always remember that there is no 'user', only the tool and it is inherently stupid! See UNATTENDED TEST EXECUTION
- plan for possible incidents because otherwise the tool just stops in its tracks or races along trying to drive the application even if it is still waiting for some input and doesn't react (EXPECT INCIDENTS)
- remember that the automated tests not only drive but also react to the System under Test (SUT). If the SUT takes different times to execute the same action (depending on the data, the database, the network etc.) the automated test must be able to wait a longer or shorter time (VARIABLE DELAYS)
- make sure that tests can be executed at any time and on any machine and eventually in parallel (INDEPENDENT TEST CASES, DEDICATED RESOURCES, PRIORITIZE TESTS, PARALLELIZE TESTS)
- write logs with infos, warnings or error messages to show how a test is doing (READABLE REPORTS, EASY TO DEBUG FAILURES)
- PAIR UP
- refactor not only the code but also the data and the expected results (REFACTOR THE TESTWARE)
- Test Manager

The most important thing to learn about test automation is the difference between managing testing and test automation

A test manager is responsible for:

- the master test plan
- defining standards and processes
- developing the specific test plans and having them performed
- giving information about the test status (how many tests have been executed? how many bugs have been found (and how serious)? how many have been corrected (or not)? how much time and resources are needed? etc.)
- helping decide if the application is ready for release
- recruiting and training testers

A test manager in charge of test automation is also responsible for:

- defining the scope of test automation (SET CLEAR GOALS, DO A PILOT)
- defining standards and processes for test automation (SET STANDARDS, GOOD DEVELOPMENT PROCESS)
- defining the strategy for test automation (TESTWARE ARCHITECTURE, ABSTRACTION LEVELS, MAINTAINABLE TESTWARE)
- deciding what to automate and what not (AUTOMATE WHAT'S NEEDED, KNOW WHEN TO STOP)
- provide the resources, people, machines and tools (DEDICATED RESOURCES, DEPUTY, RIGHT TOOLS, PLAN SUPPORT ACTIVITIES)
- planning development and maintenance of test automation (SHORT ITERATIONS, WHOLE TEAM APPROACH, MAINTAIN THE TESTWARE, REFACTOR THE TESTWARE)
- having the automated tests executed, eventually selecting an appropriate subset to run (UNATTENDED TEST EXECUTION, PRIORITIZE TESTS, TEST SELECTOR)
- recruiting and training test automators (AUTOMATION ROLES, GET TRAINING)
- getting MANAGEMENT SUPPORT for test automation by communicating with management, testers and development (SHARE INFORMATION, CELEBRATE SUCCESS, SELL THE BENEFITS)

Most recommended:

- CHECK-TO-LEARN: Let new team members learn by checking the health of the existing automation testware.
- DOCUMENT THE TESTWARE: Use this pattern from the very beginning of the automation effort.
- DO A PILOT: use this pattern when the automation effort is starting, so that team members can learn hands on.
- GET TRAINING: this pattern is the right choice after tools and architecture have been selected, but also for testers to learn how to write test cases suitable for automation.
- ASK FOR HELP: This pattern is a no-brainer! Use it!

Other useful patterns:

- PAIR UP: This is the pattern of choice when newbies are supposed to integrate into the team as fast as possible.
- TAKE SMALL STEPS: Use this pattern to learn, you don't have to build Rome in one day.
- STEEL THREAD: Good pattern for learning to navigate test automation through the SUT.

- PREFER FAMILIAR SOLUTIONS: use this pattern if team members already use tools or processes that can be applied successfully also to the test automation project.

3.1.34. LITTER BUG (*Execution Issue*)

Issue Summary

Tests leave spurious data in databases, history etc. that slows performance or hinders further tests

Category

Execution

Examples

1. A test changes the database so that it cannot run again with the same initial conditions.
2. If a test fails, it leaves the System under Test in such a condition that no more tests can run.
3. A test writes so much stuff that the next tests either cannot execute or are much slower.

Questions

Are your tests Independent from each other?

What happens when the system crashes?

Resolving Patterns

- FAIL GRACEFULLY: If a test fails it should restore the system and the environment so that the successive tests are not affected.

3.1.35. LOCALISED REGIMES (*Management Issue*)

Issue Summary

Tool use or testware architecture is different from team to team

This issue is closely related to the failure pattern FRAMEWORK COMPETITION from Michael Stahl

Category

Management

Examples

1. "Everyone will do the sensible thing": most will do something sensible, but different.

2. "Use the tool however it best suits you": ignores cost of learning how best to automate.
3. Effort is wasted by repeatedly solving the same problem in different ways
4. No re-use between teams.
5. Multiple learning curves.

Questions

Is there an overall strategy for automation?

Is there any person charged with coordinating automation for the company/enterprise?

Resolving Patterns

Most recommended:

- DESIGN FOR REUSE: Design reusable testware.
- DON'T REINVENT THE WHEEL: Use available know-how, tools and processes whenever possible.
- SET STANDARDS: Set and follow standards for the automation artefacts.
- TEST AUTOMATION OWNER: Appoint an owner for the test automation effort.

Other useful patterns:

- GET TRAINING: Plan to get training for all those involved in the test automation project.
- SHARE INFORMATION: Ask for and give information to managers, developers, other testers and customers.

3.1.36. LONG SET-UP (Design Issue)**Issue summary**

Set-up of the initial conditions for the test cases is long and complicated

Category

Design

Examples

1. The initial conditions consist in MS Access databases. Since the structure of the tables can change with every release the original databases have to be updated practically before each run. The update can take hours!
2. The insertion of data in the initially empty databases is rendered difficult by all kinds of triggers.

Questions

Do the initial conditions have to be set new for every test? Can they be shared between different tests?

Can the initial conditions be prepared outside of the automated testing suite?
Are there tools available to allow conditions/data sets to be prepared outside of the automation suite?

Resolving Patterns

Most recommended:

- CHAINED TESTS: use this pattern to have previous tests prepare the initial conditions for the following ones.
- SHARED SETUP: use this pattern to run the setup only once.

3.1.37. MANUAL INTERVENTIONS (*Execution Issue*)

Issue summary

Test automation needs lots of manual handlings to work.

Category

Execution

Examples

1. The initial conditions for a test suite have to be set up by hand.
2. In some tests there are windows or components that could not be automated and have to be driven by hand.
3. Execution times vary greatly and thus test execution is stopped and restarted manually
4. In order to run a retest all the other tests in the test suite have to be inactivated manually.
5. Test automation is started manually.
6. Automated verification is only simple so you can't tell if a test has passed or failed without looking at it.
7. Reporting of test results requires manual effort.
8. Integration of tools requires manual effort, and is slow, costly and unreliable.

Questions

Does the current test automation tool drive the Software Under Test (SUT) reliably?
Are the initial conditions set automatically? If not, why not?

Resolving Patterns

Most recommended:

- UNATTENDED TEST EXECUTION: Use this pattern for issues like Examples 1. and 5.
- ONE-CLICK RETEST: this pattern helps solve issues similar to Example 4.
- VARIABLE DELAYS: use this pattern if you have an issue like Example 3.

The following patterns will help you solve issues like Example 2.

- **MIX APPROACHES:** Don't be afraid to use different tools or approaches.
- **RIGHT TOOLS:** Select the right tools for your application, environment, people, budget and level of testing.
- **TESTABLE SOFTWARE:** Locate what kind of implementations in the Software Under Test (SUT) may make test automation difficult or impossible and find a solution as early as possible.

Better tool integration will help to solve issue 7. You may need to write some code to coordinate between the tools, an "automated bridge", but this will result in seamless automation rather than manual automation.

3.1.38. *MANUAL MIMICRY (Design Issue)*

Issue Summary

Automation mimics manual tests without searching for more efficient solutions

Category

Design

Examples

The story, which we have from Michael Stahl who called this issue the Sorcerer's Apprentice Syndrome, goes as following:

Everyone, I assume, is familiar with Disney's Fantasia. The piece about the Sorcerer's Apprentice is probably the best-known part. Let's look at what happens in this scene with professional eyes:

Mickey is assigned with a repetitive, boring task, he has to carry water from a big well to a smaller pool located many steps lower. After doing it for a while, he figures out that this problem can be solved by Automation. He takes a broom, and quickly writes a script in his favorite programming language to make the broom execute the job automatically. The design of the automated system mimics exactly the actions Mickey would use to perform the same task. Two hands, two buckets, walk to the well, fill the buckets, walk to the destination pool, empty buckets.

What's this got to do with automation?

This is a common occurrence in test automation: manual test cases are taken step by step, and each step is translated to code that performs the exact same action. And here lies the mistake. Mimicking human actions sounds straight forward but is sometimes hard to accomplish programmatically and is frequently inefficient. Consider the problem Mickey is faced with: "Transfer water from one location to another; The water well is at higher elevation than the destination pool". Mickey's solution is mechanically unstable and complex: tall, unbalanced structure; mechanical arms that move in a number of direction and must support the weight of the water; ability to go up and down stairs. Ask a mechanical engineer – building this machine is pretty difficult; some magic would probably help. Compare it to the trivial solution: a pipe and gravity.

However, arriving to this efficient solution means departing from the written test steps – which calls for an ability to distance oneself from the immediate task.

Many automation solutions – definitely those that started small and mushroomed - suffer from this problem. The step-by-step conversion of a manual test to an automated one results in inefficient, complex and brittle test automation system.

Derek Bergin explains further:

The automation team often tackles technical debt by just blindly automating the manual test suite

Most manual test suites have evolved over time with the development of the product under test. Rarely, if ever, are they refactored to remove redundancy. Furthermore, even a well-designed manual test is planned around the time and boredom limitations of a human. Just simply taking these test cases and automating them is wasting a huge opportunity to use automation properly.

The existing tests should be examined for their purpose and what validation criteria are being used. Evaluate the tests for order of automation. My preference is to use 'amount of tester time saved' as a primary indicator of 'value' and thus allow the warm brains to focus on the things they do best – like exploratory testing.

Comments from Dot:

Trying to "automate all manual tests" is a mistake in two ways:

1. Not all manual tests should be automated! Tests that take a long time to automate and are not run often, tests for usability issues (do the colours look nice? is this the way the users will do it?), and some technical aspects (e.g. captcha) are better as manual tests.
2. If you automate ONLY your manual tests, you are missing some important benefits of automation (as Derek mentions). This includes additional verification, ways of testing other values around a central test point, and some new forms of automated testing using pseudo-random input generation and heuristic oracles.

Questions

Who designs the test cases to be automated? Are the automated tests just a copy of the manual test?

Do the automators "understand" the application they are automating? Can they see ways of achieving their goals with automation that might differ from the way a manual test would be run?

Have different ways of organising the automated tests been considered, taking advantage of things that are easier to do with a computer than with human testers? e.g. longer test runs but with many short independent tests)

Have the tests considered additional verification that could be done with automated tests that would be difficult or impossible with manual tests? (e.g. checking the state of a GUI object "behind the scenes")

Resolving Patterns

Most recommended:

- **KEEP IT SIMPLE:** Use the simplest solution you can imagine.
- **KEYWORD-DRIVEN TESTING:** Tests are driven by keywords that represent actions of a test, and that include input data and expected results.
- **LAZY AUTOMATOR:** Lazy people are the best automation engineers.
- **ONE CLEAR PURPOSE:** Each test has only one clear purpose.
- **THINK OUT-OF-THE-BOX:** try to look at the problem from unusual viewpoints.

Other useful patterns:

- **DOMAIN-DRIVEN TESTING:** Develop a domain-specific language for testers to write their automated test cases with.

A related issue is *INTERDEPENDENT TEST CASES* where tests depend on the results of previous tests.

3.1.39. *MULTIPLE PLATFORMS (Design Issue)*

Issue Summary

The same tests are supposed to run in many different operating systems or browsers

Category

Design

Examples

1. The tests have to run with different database systems (for instance Oracle, MySQL etc.).
2. The tests have to run on different browsers (for instance Internet Explorer, Firefox, Chrome etc.).

Questions

Can you list exactly on what systems the tests are supposed to run?

Is the test execution really identical?

Resolving Patterns

Most recommended:

- **ABSTRACTION LEVELS:** use this pattern to isolate the technical implementation. In this way you will be able to keep platform differences to a minimum.
- **GET ON THE CLOUD:** use this pattern to implement efficiently different platforms.

- THINK OUT-OF-THE-BOX: try to look at the problem from unusual viewpoints.

Other useful patterns:

- TEST AUTOMATION FRAMEWORK: this pattern will help you implement other patterns, for instance ABSTRACTION LEVELS.

3.1.40. NO INFO ON CHANGES (Process Issue)

Issue summary

Development changes are not communicated to the test automators, or not in good time.

Category

Process

Examples

The following changes in the SUT can have very negative impact on the current automation if they are not communicated timely:

1. new development environment
2. new programming language
3. new operating system
4. new GUI-components
5. sequence of the windows
6. database structure
7. database triggers

Questions

Who is in charge of the changes?

Who should be notified? Do the developers know it?

Resolving Patterns

- SHARE INFORMATION: apply this pattern to get rid of this issue once and for all.
- SHORT ITERATIONS: With short feedback-cycles eventual problems can be found out earlier and more efficiently.
- WHOLE TEAM APPROACH: use this pattern if you use an agile development process. If you haven't gone agile yet, check if it would improve development (and of course also test automation) in your context.

3.1.41. NO PREVIOUS TEST AUTOMATION (*Management issue*)

Issue summary

You are supposed to start automating tests, but neither you nor your team has any experience in test automation and it hasn't ever been implemented in the company, or it was tried earlier and failed so you need to start again.

Category

Management

Examples

1. You have never successfully got going with test automation.
2. You have tried in the past with test automation, but it was abandoned.
3. You are wanting to automate in a way that you haven't done before (e.g. GUI level system testing if you have automated unit tests already).

Questions

What are the reasons for wanting to automate testing now?

What measurable objectives will determine success for the automation?

What resources and support will be given to this effort?

If automation has failed in the past, analyse what happened. Were the main reasons due to management issues or technical issues?

Talk to people who were around at the time. What would they do differently? What advice can they give you?

Resolving Patterns

We recommend doing these patterns in this order:

1. SET CLEAR GOALS: This pattern is critical. It must be applied at the beginning of any big or small automation effort.
2. MANAGEMENT SUPPORT: Also critical. Automation efforts that are driven only by a lone hero are much more apt to stall because nobody else knows about it or can use and maintain what has been developed.
3. TEST AUTOMATION OWNER: appoint somebody not only to introduce test automation, but to keep it up and running also in the future.
4. DEDICATED RESOURCES: This pattern is especially important at the beginning of a new automation effort. Depending on the size of your automation you can later slacken its use.
5. RIGHT TOOLS: This pattern is essential not only for long lasting automation, but also for quick fixes.
6. AUTOMATION ROLES: Use this pattern to fill the roles you need to develop the automation testware. If possible use a WHOLE TEAM APPROACH and if needed GET TRAINING for your people.
7. PLAN SUPPORT ACTIVITIES: Don't forget to apply this pattern if you want to be able to keep your schedules. Missing support from specialists can ground a project pretty effectively!

8. MAINTAINABLE TESTWARE: Apply this pattern from the very beginning if you want your automation effort to be long lasting and your maintenance costs low.
9. AUTOMATE WHAT'S NEEDED: This pattern shows you how to select the features most worthy to be automated.
10. TAKE SMALL STEPS: This pattern is especially important in the beginning, but it should always be kept in mind.
11. UNATTENDED TEST EXECUTION: This pattern gives you the last suggestions on how to get done with test automation.

3.1.42. NON-TECHNICAL-TESTERS (Process Issue)

Issue summary

Testers aren't able to write automation test cases if they are not adept with the automation tools.

Category

Process

Examples

1. Testers are interested in testing and not all testers want to learn the scripting languages of different automation tools. On the other hand, automators aren't necessarily well acquainted with the application, so there are often communication problems.
2. Testers can prepare test cases from the requirements and can therefore start even before the application has been developed. Automators must usually wait for at least a rudimentary GUI.

Resolving Patterns

Most recommended:

- DOMAIN-DRIVEN TESTING: Apply this pattern to get rid of this issue for sure. It helps you find the best architecture when the testers cannot also be automators.
- OBJECT MAP: This pattern is useful even if you don't implement DOMAIN-DRIVEN TESTING because it forces the development of more readable scripts.

Other useful patterns:

- KEYWORD-DRIVEN TESTING: This pattern is widely used already, so it will be not only easy to apply for your testers, but you will also find it easier to find automators able to implement it.
- SHARE INFORMATION: If you have issues like Example 1. this is the pattern for you!

- TEST AUTOMATION FRAMEWORK: If you plan to implement DOMAIN-DRIVEN TESTING you will need this pattern too. Even if you don't, this pattern can make it easier for testers to use and help implement the automation.

3.1.43. OBSCURE MANAGEMENT REPORTS (Management Issue)

Issue summary

Reports from test automation are not very useful to monitor the process and keep management informed

Category

Management

Examples

1. Test automation reports only if the current tests passed or failed, trends have to be built manually.
2. It is difficult to get a "big picture" about what parts of the system are OK or have serious problems.

Questions

What kind of information is currently delivered by test automation?

What kind of information does management need?

Resolving Patterns

Most recommended:

- READABLE REPORTS: apply this pattern to give your stakeholders the information they need.

Other useful patterns:

- LEARN FROM MISTAKES: apply this pattern to learn to do better next time!
- SHARE INFORMATION: apply this pattern in order to find out what kind of reports are needed.

3.1.44. OBSCURE TESTS (Design Issue)

Issue summary

Automated tests are very complex and difficult to understand.

Category

Design

Examples

1. Manual testcases have been automated as is, that is they depict long scenarios that build on each other.
2. Automated scripts or data are not modular, are not documented and are stored with cryptical names, so that they practically cannot be reused.
3. Automation scripts are written in the proprietary language of the current tool and testers cannot read and much less run them.
4. Keywords have been implemented, but never maintained, so that nobody really knows what they do.
5. The automated test cases try to test too much all at the same time. This makes the tests not only slower, but also more fragile because they can fail in parts that actually have nothing to do with the original test.

Questions

Who is doing the automation? A "lone hero"?

Who is writing the automation test cases?

Are there standards? If yes, are they used?

Resolving Patterns

Most recommended:

- DOCUMENT THE TESTWARE: Use this pattern for issues similar to Examples 2. 3. or 4.
- DOMAIN-DRIVEN TESTING: use this pattern to enable even non-technical testers to write and review test cases for automation.
- GOOD DEVELOPMENT PROCESS, GOOD PROGRAMMING PRACTICES: these two patterns ensure that you get rid of issues similar to Example 2.
- MAINTAINABLE TESTWARE: Use this pattern to develop automation that can be used not only by the original developer.
- ONE CLEAR PURPOSE: use this pattern if your issue is similar to Examples 1. or 5.
- REFACTOR THE TESTWARE: "bite the bullet" and take the time to re-organise the tests - it will save you time in the long run.
- SET STANDARDS: this pattern helps you find a common language between all team members.
- SINGLE PAGE SCRIPTS: use this pattern to modularize your scripts.
- TOOL INDEPENDENCE: use this pattern if you don't want everybody on the team to become a tool specialist.

Other useful patterns:

- DESIGN FOR REUSE: using this pattern forces the team to develop readable and maintainable testware.
- INDEPENDENT TEST CASES: apply this pattern to get rid of issues like Example 1.

- KEEP IT SIMPLE: this pattern is always recommendable!
- KILL THE ZOMBIES: this pattern helps you clean up your scripts.
- LEARN FROM MISTAKES: apply this pattern in order to transform your bad experiences in learning experiences.
- MAINTAIN THE TESTWARE: this pattern is necessary as soon as you want to use your automation for more than a quick fix.
- OBJECT MAP: This pattern helps you develop scripts that use readable names.

3.1.45. REPETITIOUS TESTS (*Design Issue*)

Issue summary

Test cases repeat the same actions on different data

Category

Design

Examples

1. Test cases have the same structure, for instance entering a customer with his address, but use different values.

Questions

Are the test cases really similar?

Resolving Patterns

Most recommended:

- DATA-DRIVEN TESTING: this pattern is one of the most applied on the field (even if nobody knows that it's a pattern!).

Other useful patterns:

- DON'T REINVENT THE WHEEL: applying this pattern will help you develop reusable testware.
- REFACTOR THE TESTWARE: re-design the tests so that they are more efficient.

3.1.46. SCHEDULE SLIP (*Management Issue*)

Issue Summary

The planned automation is not keeping up with its schedule for developing the automation or for automating tests.

Category

Management

Examples

1. Test automation is done only in people's spare time
2. Team members are working on concurrent tasks which take priority over automation tasks
3. Schedules for what automation can be done were too optimistic
4. Necessary software or hardware is not available on time or has to be shared with other projects
5. The planned schedule was not realistic

Questions

What is the priority for test automation?

Is test automation supported by management or is it performed by a "lone crusader"?

Who is doing the planning for test automation? How is the required time for automation calculated?

What software or hardware is needed? When?

Resolving Patterns

Most recommended:

- SHORT ITERATIONS: using this pattern you will be able to deliver something sooner, even if not what you had originally hoped or planned.
- WHOLE TEAM APPROACH: if development is using an agile process, then this is the pattern to apply.
- DEDICATED RESOURCES: this pattern will help you pick up speed.
- PLAN SUPPORT ACTIVITIES: use this pattern from the beginning so that resources are available when needed.

Other useful patterns:

- FULL TIME JOB: this pattern helps the automation team work more efficiently.
- MANAGEMENT SUPPORT: use this pattern to free DEDICATED RESOURCES.

3.1.47. SCRIPT CREEP (Process issue)

Issue Summary

There are too many scripts and it is not clear if they are still in use or not.

Category

Process

Examples

1. It takes so much time to check if a script is already available that testers or automators would rather write a new one instead. This means that there are a lot of very similar scripts.
2. Nobody "refactors" the scripts so that after a time some fail consistently and are not executed any longer.
3. It isn't possible to check which scripts are actually in use.

Questions

How are scripts documented?

Are there standards regarding naming and documentation?

Who writes the scripts? Who uses them?

Is anyone charged with reviewing the relevance and usefulness of the scripts at regular intervals?

Resolving Patterns

Most recommended:

- GOOD PROGRAMMING PRACTICES: you should have been using this pattern from the very beginning.
- MAINTAINABLE TESTWARE: this pattern will tell you how to avoid such problems in future.
- MAINTAIN THE TESTWARE: you should already use this pattern.
- MANAGEMENT SUPPORT: You will need this pattern to be able to change the current bad behaviour.
- REFACTOR THE TESTWARE: you should already use this pattern.

You should already be applying these patterns. If not, do it!

Other useful Patterns:

- GOOD DEVELOPMENT PROCESS: apply this pattern if you don't have a process for developing test automation. Apply it also if your process lives only on paper (nobody cares).
- LEARN FROM MISTAKES: apply this pattern to turn mistakes into useful experiences.
- KILL THE ZOMBIES: Apply this pattern for a start.
- DOCUMENT THE TESTWARE: you should be already applying this pattern. Retrofixing documentation is quite an effort. Do it in the future for all new projects and everytime you have to update something old.
- KEEP IT SIMPLE: Always apply this pattern!
- SET STANDARDS: apply this pattern so that team members "speak the same language" and hence can efficiently share their work.

3.1.48. STALLED AUTOMATION (*Process Issue*)

Issue Summary

Automation has been tried, but it never "got off the ground" or has now fallen into disuse.

Category

Process

Examples

1. The automation tools were tried but are not used efficiently or have already become shelf-ware.
2. The current tools are no longer supported or don't fit the current version of the Software Under Test (SUT).
3. The automation has been implemented by a lone champion without any support from management and he or she has left the company.
4. Testers are expected to do the test automation, but cannot use the tools efficiently (see *NON-TECHNICAL-TESTERS*).
5. You wanted to automate everything and got stuck with not automatable features.
6. The SUT changes so fast that you just have time to fix the current automation but have no resources to do new stuff.
7. In order to automate new functionalities, you need resources or specialists that are not available.
8. No one is adding new automated tests, so only what already exists is used, and no one is looking at it.
9. Existing tests are being run, but if there are problems with a test, it is more likely to be removed than the problem to be fixed.

Questions

Who is doing the automation?

Is someone coordinating the automation efforts or is everyone on their own?

Who has selected the tools and how?

Are the current tools being used? If not, why not?

Has all existing and previous work in automation been documented?

What has changed since the automation was originally developed?

Can you quantify the benefit of investing resources (time, effort) into the automation?

Resolving Patterns

Most recommended:

- DO A PILOT: Use this pattern to find out why the automation effort got stalled and what to do about it.
- LAZY AUTOMATOR: This is the right pattern for a lone champion!
- MAINTAIN THE TESTWARE: You should already be using this pattern. Not using it could be why your automation effort has stalled.

- TEST AUTOMATION OWNER: appoint an "owner". If nobody feels ownership for test automation, nobody will really care if it's successful or not.
- WHOLE TEAM APPROACH: Use this pattern if you want your test automation effort to be effective and successful. It will be easy to implement if the development team for the SUT already adopted an agile process. Otherwise you will have to convince development to do so before you can apply this pattern.

Other useful Patterns:

- ABSTRACTION LEVELS: use this pattern if your issue is similar to Example 4.
- KNOW WHEN TO STOP: This pattern helps you recognize that you cannot automate everything (see Example 5).
- LEARN FROM MISTAKES: this pattern helps turn mistakes into learning experiences.
- MANAGEMENT SUPPORT: Apply this pattern to get the support you need in order to use other patterns.
- SELL THE BENEFITS: If your issue is similar to Example 3. and nobody is supporting you, this pattern can help you get the support you need.
- TAKE SMALL STEPS: Apply this pattern to get going again with the automation effort.
- TEST AUTOMATION FRAMEWORK: Check if your problems would be solved with this pattern.

3.1.49. SUT REMAKE (Management Issue)

Issue Summary

Substantial changes are being made to the SUT or it is to be rewritten from scratch

Category

Management

Examples

Changes to the SUT can be for instance:

1. New development technology.
2. Substitution of known components with new ones.
3. Changed navigation.
4. Changed database structure.
5. Localization.

Questions

Are the current tools still usable?

Does development communicate what is being changed?

Resolving Patterns

Most recommended:

- DO A PILOT: Use this pattern to find out if the current tools and automation strategies are still valid (recommended if your issue is similar to Example 1).
- WHOLE TEAM APPROACH: If you have to automate a completely new SUT (Example 1) and the development team has adopted an agile process this is the pattern of choice.
- ABSTRACTION LEVELS: if you have to substantially rework your scripts, use this pattern on all the scripts you have to touch: you will have much less work when such changes happen again.

Other useful Patterns:

- LEARN FROM MISTAKES: this pattern helps you learn from your current experience how to do better.
- MANAGEMENT SUPPORT: Apply this pattern to get the support you need in order to use other patterns.
- SHARE INFORMATION to learn what will be changed and to inform development when changes disrupt automation scripts.

3.1.50. TEST DATA LOSS (Process Issue)

Issue Summary

Test data isn't secured so that it has to be generated again and again.

Category

Process

Examples

1. Testers create new data every time because it's too difficult to check if test data is already available.
2. Every tester has his own data portfolio and doesn't share it with test automation.
3. Anonymized production data is used and it has to be generated anew every time.

Questions

How is test data structured? Text files? Databases? Other?

Are there naming and documentation standards?

Who creates the data? Who uses it?

Resolving Patterns

Most recommended:

- **LAZY AUTOMATOR:** Apply this pattern if you want to resolve the issue once and for all. It contains all the other resolving patterns!

Other useful patterns:

- **DESIGN FOR REUSE:** This pattern should have been applied from the very beginning of the automation effort. Still, if you haven't, do it for all new automation projects. Use it also every time you have to refactor something.
- **DON'T REINVENT THE WHEEL:** use this pattern as often as possible!
- **GOOD DEVELOPMENT PROCESS:** Again, a pattern that should have been applied from the start. If you didn't, do so now!
- **GOOD PROGRAMMING PRACTICES:** same as for GOOD DEVELOPMENT PROCESS!

3.1.51. TOO EARLY AUTOMATION (*Design Issue*)

Issue summary

Test automation starts too early on an immature application or on the wrong aspect of an application and produces only "noise".

Category

Design

Examples

1. Since the SUT is not stable yet, you have to update the automation so often that you don't make any progress. You are always only tinkering with the same scripts.
2. Management has suspended development for the SUT. All your work was a waste of time.
3. The aspects you are trying to automate keep changing and the automation can't cope with it.

Questions

How stable is the SUT? Is there some part that is already "testable"?

How about the test cases? Do they change just as often?

Are you trying to automate aspects that are least stable, while other aspects could more usefully be automated?

For example, trying to automate tests through a GUI when the GUI design is not yet stable is asking for trouble. However, it may not be too early to automate tests through the API or at unit test level, or to put in the foundations for tests to be written later on using a high-level abstraction approach.

Resolving Patterns

Most recommended:

- WHOLE TEAM APPROACH: if development already uses an agile process then this is the pattern of choice to solve this issue.
- If the instability is due to superficial aspects, rather than fundamental design issues, you could SHARE INFORMATION about the impact of these changes on the automation, and possibly use MANAGEMENT SUPPORT in order to stabilise unnecessary changes.
- ABSTRACTION LEVELS: it may be too early to automate at the user interface level, but you may be able to design your TESTWARE ARCHITECTURE so that there are useful and long-lasting scripts you can develop now. Also, you can devise a way for tests to be written even if the user interface is not yet stable, using DOMAIN-DRIVEN TESTING and/or KEYWORD-DRIVEN TESTING.

Other useful patterns:

- AUTOMATE GOOD TESTS: this pattern helps you select what to automate.
- DO A PILOT: this pattern can be used to explore which aspects would be best to automate, and what should be automated early and what is better left until later.

3.1.52. TOOL DEPENDENCY (Design Issue)

Issue summary

Test automation is strongly dependent on some special tool.

Category

Design

Examples

1. You have written all the automation scripts in the proprietary language of a particular capture-replay tool.
2. You have to change to a different tool, but now you can't run any of your existing scripts without major work.

Questions

How to make sure that if and when the tool will have to be changed, you don't have to start again from scratch?

Resolving Patterns

Most recommended:

- RIGHT TOOLS: use this pattern if your present tool doesn't fit the Software Under Test (SUT) well enough.
- TOOL INDEPENDENCE: this is the pattern of choice to solve this issue.

Other useful patterns:

- ABSTRACTION LEVELS: this pattern helps you separate the technical tool layer from the functional test cases.
- OBJECT MAP: use this pattern with all tools.

3.1.53. TOOL-DRIVEN AUTOMATION (*Process Issue*)

Issue Summary

Test cases are automated using “as is” the features of a test automation tool

Category

Process

Examples

1. In older tools this means using the capture functionality to develop test cases: while a test case is executed manually the tool records all the tester actions in a proprietary script. By replaying the script, the test case can be executed again automatically (look up the pattern CAPTURE-REPLAY for more details).
2. In more modern tools you can record "keyword" scripts, but the mode of operation is actually much the same as in capture-replay.
3. Both approaches harbour serious problems: The speed and ease in recording test cases can induce testers to record more and more tests (keyword scripts), without considering GOOD PROGRAMMING PRACTICES such as modularity, setting standards and so on that enhance reuse and thus maintainability. Without reuse, the effort to update the automation in parallel with the changes in the SUT (System under Test) becomes more and more grievous until you end up with STALLED AUTOMATION.

Resolving Patterns

Most recommended:

- TOOL INDEPENDENCE: Separate the technical implementation that is specific for the tool from the functional implementation.
- GOOD PROGRAMMING PRACTICES: Use the same good programming practices for test code as in software development for production code.
- LAZY AUTOMATOR: Lazy people are the best automation engineers.
- TEST AUTOMATION OWNER: Appoint an owner for the test automation effort.
- TESTWARE ARCHITECTURE: Design the structure of your testware so that your automators and testers can work as efficiently as possible.
- WHOLE TEAM APPROACH: Testers, coders and other roles work together on one team to develop test automation along with production code.

Other useful patterns:

- LOOK FOR TROUBLE: Keep an eye on possible problems in order to solve them before they become unmanageable.
- MAINTAINABLE TESTWARE: Design your testware so that it does not have to be updated for every little change in the Software Under Test (SUT).
- SET STANDARDS: Set and follow standards for the automation artefacts.
- GET TRAINING: Plan to get training for all those involved in the test automation project.

3.1.54. UNAUTOMATABLE TEST CASES (*Design Issue*)

Issue summary

Existing test cases are “unautomatable”.

Category

Design

Examples

1. Test cases are written in a very "terse" way assuming that manual testers know the SUT inside out. Automators usually cannot understand what they are supposed to automate.
2. Test cases build on each other to complex scenarios that make it very difficult to write test cases independent from each other.
3. Test cases are performed only once in a while and are quite complicated.

Questions

Who writes the test cases?

Has time been planned to explain the test cases to the automation team?

Resolving Patterns

Most recommended:

- SHARE INFORMATION: apply this pattern to clear misunderstandings between testers and automators.
- WHOLE TEAM APPROACH: apply this pattern to get rid of misunderstandings in the team.
- THINK OUT-OF-THE-BOX: try to look at the problem from unusual viewpoints.

Other useful patterns:

- KNOW WHEN TO STOP: this pattern helps to recognize that not all tests are automatable.

- MANAGEMENT SUPPORT: you will need to use this pattern especially if you want to implement a WHOLE TEAM APPROACH and nobody is going agile yet in your company.
- PAIR UP: apply this pattern to enhance communication and learning between testers and automators.

3.1.55. UNFOCUSED AUTOMATION (*Process Issue*)

Issue Summary

What to automate is selected ad-hoc.

Category

Process

Examples

1. Parts of the SUT that would deliver the most ROI are either not automated at all or are automated after parts that are not so important.
2. Automators implement tests that are easy to automate, instead of asking the testers which tests are most important to automate.
3. Only "strategic" applications are automated, which means there still is a lot of tedious manual testing.
4. A good automation project gets stopped arbitrarily.

Questions

Who selects what to automate?

Who does the automation?

What are the most important tests to automate (first)?

Resolving Patterns

Most recommended:

- AUTOMATE WHAT'S NEEDED: Apply this pattern to solve this issue.
- WHOLE TEAM APPROACH: If you are just starting with test automation and the SUT is being developed using agile processes, then this is the pattern of choice, because the whole team can best decide what to automate. Otherwise try first to convince development to adopt an agile process (you will definitely need MANAGEMENT SUPPORT).

Other useful patterns:

- KNOW WHEN TO STOP: apply this pattern if you don't want to waste a lot of effort trying to automate test cases that should not be automated.
- LEARN FROM MISTAKES: this pattern helps you turn bad experiences into learning experiences.

3.1.56. *UNMOTIVATED TEAM (Management Issue)*

Issue summary

The test automation team is not motivated.

Category

Management

Examples

1. Automation team members resist switching to a different better tool than the one they have been using.
2. Team members don't collaborate and are not willing to share their know-how by pairing with newbies.
3. Testers don't want to help because they have had too many bad experiences with test automation.
4. Test automation team members are frustrated because:
 - they have no time to develop new test automation and are always only tinkering with the old scripts
 - development changes the SUT-GUI all the time and they find out only when the tests fail at run time
 - they need help from specialists and nobody has time
 - due to technical problems they need too much time to implement new test automation

Questions

Are all team members unmotivated or only some?

Are the people on the test automation team by choice? Or were they ordered to it by management?

Does the test automation team get the support that it needs?

Resolving Patterns

Most recommended:

- CELEBRATE SUCCESS: to motivate the team use this pattern everytime some milestone has been reached.
- GET TRAINING: use this pattern if your issue is similar to Example 1.
- WHOLE TEAM APPROACH: if the development team uses an agile process, then this pattern will help even unsatisfied people to better integrate (Example 2).

Other useful patterns:

- PLAN SUPPORT ACTIVITIES: Apply this pattern if your team is frustrated for reasons similar to Example 4.
- SHARE INFORMATION: Apply this pattern if frustration is mainly due to poor communication.
- SHORT ITERATIONS: use this pattern to get shorter feedback cycles, so you notice sooner when somebody isn't motivated.

- LEARN FROM MISTAKES: use this pattern if frustration is due to bad experiences.

3.1.57. UNREALISTIC EXPECTATIONS (*Management issue*)

Issue summary

There are unrealistic expectations regarding what test automation can and cannot deliver.

Usually it is managers who have unrealistic expectations for automation - sometimes they are hoping for a "silver bullet" that will solve all of their problems. However, others may also have unrealistic expectations. For example, testers may believe that automated tests will "notice" things that a human being would notice, or developers may think that tests will "automate themselves" and testers are not needed to identify which tests are best to automate.

Category

Management

Examples

1. Every single (manual) test case can and should be automated.
2. Even if the SUT is changed almost daily, the tests should be automated (now).
3. There is no need to spend time developing a framework - the manager bought you the tool, now just get on with it.
4. When automated tests pass, it means there are no defects in the software.
5. Running an automated test the first time automatically gives Return on Investment (even if it takes 10 times longer to automate the test than it would to run it manually).
6. Any manual tester can write automated test cases directly in the tool scripting language.
7. Maintenance of automated tests will not be needed or has a very small cost.
8. Managers don't understand that introduction of test automation introduces a parallel development process and thus increases cost.

Questions

What do managers expect that automation will do for the company?

What sources have they used to find out about automation?

What do they see as the limitations of automation?

What investment do they think is still needed in automation?

How will they measure the benefits of automation?

Resolving Patterns

Most recommended:

- **SET CLEAR GOALS:** Apply this pattern if you still have to begin. If you are already automating, take a step back and apply this pattern so long until everybody has the same understanding of test automation.
- **DO A PILOT:** When the automation effort hasn't started yet, apply this pattern to clarify what automation can and cannot do, what resources you will need (tools, people) etc.
- **SHARE INFORMATION:** This pattern is useful to resolve this issue if you are already in the thick of test automation. If people believe any of the things in the list above of unrealistic expectations, you will need to educate them to explain why this is not realistic.

Other useful patterns:

- **TAKE SMALL STEPS:** Use this pattern to show what can realistically be achieved within a short time frame such as a single sprint.

4. Part 4

4.1. Patterns in Detail

In this section, we show all of the Patterns, listed in alphabetical order, so you can find a relevant pattern easily if you know its name.

This section repeats what is in the wiki at the time of producing this book (2018). The wiki will have the most up-to-date versions of all patterns (and issues).

The wiki is much easier to use, as all other issues and patterns are links, but we wanted the book to be stand-alone, so it could be used without having access to the wiki (and the internet).

Each pattern starts with a mind map showing on the left the issues that are solved by this pattern, and on the right its relationship to other patterns.

4.1.1. ABSTRACTION LEVELS (Design Pattern)

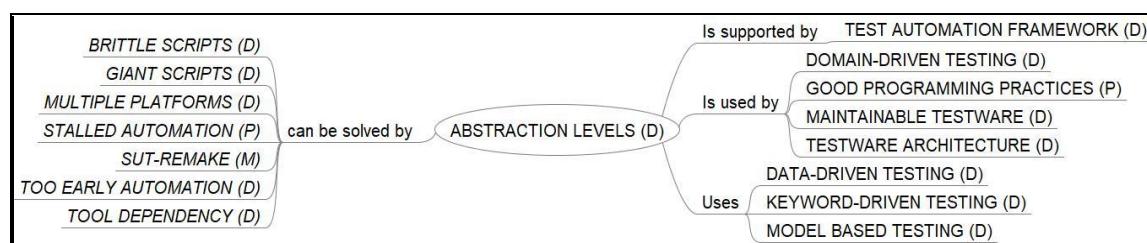


Figure 4.1.1-1 ABSTRACTION LEVELS

Pattern Summary

Build testware that has one or more abstraction layers.

Category

Design

Context

Apply this pattern for long lasting and maintainable automation. It's not necessary for disposable scripts

Description

The most effective way to build automated testware is to construct it so that it has one or more abstraction layers (or levels). For example, in software development, the code to drive the GUI is usually separated from the code that actually implements the business functionality and also separated from the code that implements access to the database. Each part communicates with the others only through an interface. In this way each can be individually changed without breaking the whole as long as the interface is not changed (this is the theory, it is not always practiced).

For test automation this means that the testware is built so that you write in the scripting language of the tool only technical implementations (for instance scripts

that drive a window or some GUI-component of the SUT). This is the lowest layer. The test cases call these scripts and add the necessary data. This is the next layer. And if you implement a kind of meta-language you get another layer. As for software code, the charm of abstraction layers is that since the layers are independent of each other they can be substituted without having to touch the other layers. The only thing that has to be maintained is the interface between them (how the test cases are supposed to call the scripts).

By separating the technical implementations in the tool's scripting language from the functional tests, you can later change automation tools with relative ease, because you will only need to rewrite the tool-specific scripts. Also if you keep the development technicalities apart from the test cases, even testers with no development knowledge will be able to write and maintain the testware. And they can start writing the tests even before the SUT has been completely developed. Another advantage is that you can reuse the technical scripts for other test automation efforts.

Implementation

There are different ways to implement abstraction layers. Which to choose depends on how evolved your test automation framework is.

- DATA-DRIVEN TESTING means that you separate the data from the execution scripts (drivers). In this case you must take care to correctly pair the data to the drivers.
- In KEYWORD-DRIVEN TESTING you specify a keyword that controls how the data is to be processed. Generally, keywords correspond to words in a domain-specific language (for instance insurance or manufacturing). It is usually used for DOMAIN-DRIVEN TESTING.
- In MODEL-BASED TESTING you create a test model of the SUT. Typically, the modelling of test sequences starts at a very abstract level and is later refined step by step. From the model, a generation tool can automatically create test cases, test data, and even executable test scripts.
- Use TOOL INDEPENDENCE to separate the technical implementation that is specific for the tool from the functional implementation.

Potential problems

Building a good TESTWARE ARCHITECTURE takes time and effort and should be planned from the beginning of an automation effort. There is a temptation (sometimes encouraged by management) to "just do it". This is fine as a way of experimenting and getting started, for example if you DO A PILOT. However, you soon find that you have many tests that are not well structured and get STALLED AUTOMATION with high maintenance costs, but now you are "locked in" to the wrong solution if you are not aware of the importance of abstraction levels.

Issues addressed by this pattern

BRITTLE SCRIPTS

GIANT SCRIPTS

MULTIPLE PLATFORMS

*STALLED AUTOMATION
SUT REMAKE
TOO EARLY AUTOMATION
TOOL DEPENDENCY*

4.1.2. ASK FOR HELP (Process Pattern)

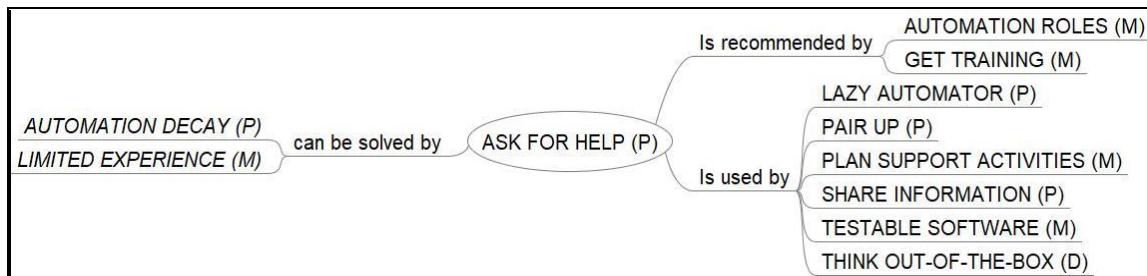


Figure 4.1.2-1 ASK FOR HELP

Pattern summary

Ask for help instead of wasting time trying to do everything yourself.

Category

Process

Context

This is an extremely important pattern if you want to do test automation efficiently: it should always be used!

Description

Ask for help when you need it. Recognise when you have got "stuck" and can't get any further on your own.

Implementation

Find out who has the expertise you are missing and ask for help. Do try to work on your own, but don't waste time. If somebody can help you to quickly solve some problem, then definitely ask.

Help may be available from other people in your organisation, but also from web sites and discussion groups and forums.

To encourage people to help you, you can use "gamification" (see experience entry from Kristoffer Nordström).

Recommendations

Don't be afraid to ask for help: most people actually enjoy helping.

Issues addressed by this pattern

*AUTOMATION DECAY
LIMITED EXPERIENCE*

4.1.3. AUTOMATE EARLY (Management Pattern)



Figure 4.1.3-1 AUTOMATE EARLY

Pattern summary

Start to automate as early as possible. The sooner automated tests can be run the sooner you can see results

Category

Management

Description

Start to automate as early as possible. The sooner you can run automated tests the sooner you can get results:

- If you work side by side with the developers from the very beginning, you will be able to see to it that the Software Under Test (SUT) is developed as TESTABLE SOFTWARE.
- The developers have feedback from the beginning.
- Management will be readier to support you if you show quickly a return on investment.

Implementation

Start to automate a STEEL THREAD or AUTOMATE WHAT'S NEEDED. If you support DATA-DRIVEN TESTING or KEYWORD-DRIVEN TESTING testers will be able to write automated test cases even before the SUT is fully implemented.

Potential problems

Beware though of TOO EARLY AUTOMATION: you would invest a lot of effort to produce only "noise"!

Issues addressed by this pattern

HIGH ROI EXPECTATIONS

4.1.4. AUTOMATE GOOD TESTS (Design Pattern)



Figure 4.1.4-1 AUTOMATE GOOD TESTS

Pattern summary

Automate only the tests that bring the most Return on Investment (ROI).

Category

Design

Context

This pattern is useful when you have to decide what to automate, so you should apply it not only when starting with test automation from scratch, but also every time you want to increase your automation

Description

Automate only the tests that bring the most return on investment. Of course, first of all you must know your test cases (remember the warning by Dorothy Graham: Automating chaos just gives faster chaos). You may need to rework them first.

Implementation

Good candidates for automation are:

- Smoke tests: since they are run very often, they are also profitable fast.
- Regression tests: as manual tests they are often boring and time-consuming and so are either performed poorly or not at all. They are well suited for automation since they usually test stable parts of the SUT and so the expected maintenance effort is reasonably small.
- Identical tests for different environments: automating such tests also pays off fast since the most effort is usually spent in automating the first environment.
- Complex tests: if manual testing is too difficult, it will not be performed.
- Tests that require machine precision.
- Repetitive, boring and time-consuming processing: set-ups for manual testing can also be automated.

Recommendations

- Remember the 80/20 rule: 20% of the application accounts for around 80% of the usage. Automate the 20% only.
- Tests that cover critical areas in the Software Under Test (SUT) should be automated first.
- Not every manual test is suitable for automation, start with the most repetitive.
- Not every regression test should be automated from the start. It makes sense to start with the less complicated and proceed later to the more difficult ones.
- Don't automate only "strategic" applications, automate also the stuff that's boring to execute and could easily be automated [1].
- If more than one team is doing automation, make sure that they don't automate the same tests.

Issues addressed by this pattern

HIGH ROI EXPECTATIONS

TOO EARLY AUTOMATION

4.1.5. AUTOMATE THE METRICS (Design Pattern)



Figure 4.1.5-1 AUTOMATE THE METRICS

Pattern summary

Automate metrics collection.

Category

Design

Context

This pattern allows you to collect metrics efficiently and reliably. If you just write disposable scripts you will not need it.

Description

By automating metrics collection, your metrics will be more reliable because they will be collected consistently and will not be so easily biased as manually collected.

Implementation

If your tool doesn't support collecting metrics, consider implementing a TEST AUTOMATION FRAMEWORK.

Some suggestions what to collect with each test run:

- Number of tests available.
- Number of tests executed.
- Number of tests passed.
- Number of tests failed (eventually classified by error severity).
- Execution time.
- Date.
- SUT Release.

You should also try to associate bug-fix information to your test run metrics. For instance:

- Number of errors removed.
- Number of errors not yet removed.
- Number of retests.
- Number of tests failed after retest.
- Average time to remove an error.

Potential problems

If possible keep track of which bugs were found with the test automation: it will help you retain support from management and testers.

Issues addressed by this pattern

INSUFFICIENT METRICS

4.1.6. AUTOMATE WHAT'S NEEDED (Process Pattern)



Figure 4.1.6-1 AUTOMATE WHAT'S NEEDED

Pattern summary

Automate what the developers or the testers need, even if it isn't tests!

Category

Process

Context

This pattern is appropriate when your automated tests will be around for a long time, but also when you just want to write one-off or disposable scripts.

Automating what isn't needed is never a good idea!

Description

Automate the tests that will give the most value. "Smoke tests" that are run every time there is any change, for example, would be good candidates for automation, as they would be run frequently and give confidence that the latest change has not destroyed everything.

When you think about test automation, the first tests that usually come to mind are regression tests that execute themselves at night or on week-ends. Actually, with automation one can support testers in many other ways, because even when testing manually (even with exploratory tests) there are lots of repetitive tasks that could be easily automated. You may need only a little script in Python or SQL to make a tester's life that much easier.

James Tony: Keep a balance between trying to test all the code paths (which is generally good) and trying to test every possible combination of inputs (which is generally bad, because it means the same lines of code get executed thousands of times, and the test suite takes so long that it doesn't get used) – the aim should be to maximize the (customer-relevant) “bugs for your buck”, i.e. the maximum number of customer-relevant issues highlighted for the smallest expenditure of time and money

Implementation

Some suggestions:

- **AUTOMATE GOOD TESTS:** Automate only the tests that bring the most Return on Investment (ROI).
- **KNOW WHEN TO STOP:** remember that not all test cases can or should be automated.
- **SHARE INFORMATION:** what do testers need? What could you deliver them? Start supporting them there.
- Try to get at least some developers "into the boat".

Good candidates for automation in addition to tests are:

- Complex set-ups: they can be easily automated and are great time savers for testers.
- DB-Data: Database-data can be automatically extracted or loaded for use in creating initial conditions or checking results. Such support is valued by developers and testers alike.

Potential problems

When people get "stuck in" to automation, they can get carried away with what can be done and may want to automate tests that aren't really important enough to automate.

Issues addressed by this pattern

INADEQUATE COMMUNICATION

NO PREVIOUS TEST AUTOMATION

UNFOCUSED AUTOMATION

4.1.7. AUTOMATION ROLES (Management Pattern)

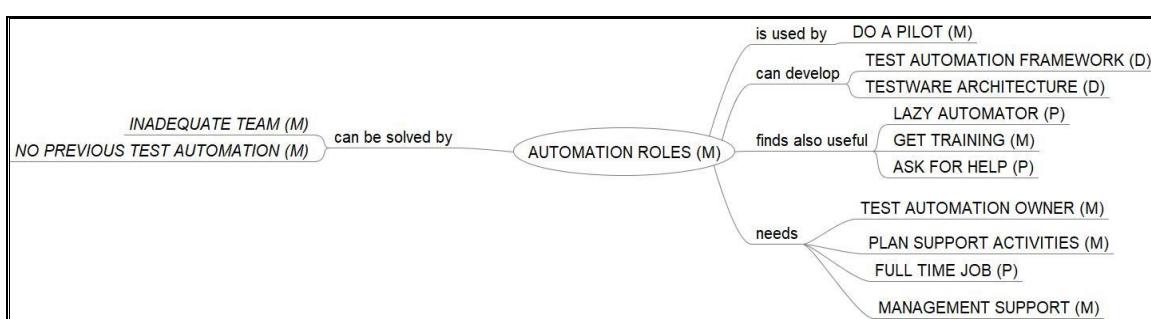


Figure 4.1.7-1 AUTOMATION ROLES

Pattern summary

Test automation needs many different skills, and people to take different roles.

Category

Management

Context

In a large organisation, there may be a team of people each with different skills and taking different automation roles.

In a small organisation, or where there is only one person currently working on automation, that person may have to take on all roles and have many different skills.

Description

There are different roles within automation, requiring different skills, ranging from the test automation architect who would design the overall TESTWARE ARCHITECTURE to a tool technical specialist, who would solve and/or report technical problems with the tool, install tool updates etc. There would also be test automators who would write and maintain scripts for testers to use.

Of course you don't necessarily need to hire new people, but then you should make sure that people GET TRAINING in whatever is appropriate for them (e.g. tool vendor courses or internal training for your own standard ways of automating your tests)

In practice you will often have team members slip into different roles, but it's important that the roles be clearly visible to the whole team.

Implementation

You need a pool of very different skills:

- Test automation engineers must have a developer skill set to be able to work with tools, do scripting and eventually to implement or maintain a TEST AUTOMATION FRAMEWORK and a LAZY AUTOMATOR mindset is definitely an extra bonus!
- Testers must have a profound knowledge of the Software under Test (SUT) and must be able to write good automated test cases but do not necessarily need to work directly with the tools.
- An automation architect must make sure that your TESTWARE ARCHITECTURE is built to be long lasting and easy to maintain.
- Depending on the SUT you may need various other specialists such as database administrators, network or security experts and so on. You should plan them to be available to help the team (PLAN SUPPORT ACTIVITIES).
- Last but not least you will need a TEST AUTOMATION OWNER, that is a champion that controls that test automation stays "healthy" and keeps an eye on new tools, techniques or processes in order to improve it.

Potential problems

Remember that skills alone don't make a good tester (or test automation engineer). Also look for the right attitude and take care that team members understand and value each other.

If you have a team of people, they may be full time on automation (FULL TIME JOB).

You may need MANAGEMENT SUPPORT to achieve this.

If you cannot have all the skills on the team all the time, try to organize a pool of experts whom you can ASK FOR HELP if required.

Issues addressed by this pattern*INADEQUATE TEAM**NO PREVIOUS TEST AUTOMATION*

4.1.8. CAPTURE-REPLAY (Design Pattern)



Figure 4.1.8-1 CAPTURE-REPLAY

Pattern summary

Capture a manual test with an appropriate tool and replay it to run the test again

Category

Design

Context

This pattern can be useful to develop one-off disposable scripts. It should not be applied for long lasting, maintainable automation, except as a stepping stone to provide very short "building block" scripts to be edited into something more useful.

Description

You can capture your manual test with an appropriate tool and replay it when you want to rerun it. Capturing a manual test can also be useful to be an "audit record" of the testing that someone has done manually, for example if a business user comes in to do some testing. A small captured script can also be useful to support a bug report.

Implementation

With a so-called capture-replay tool you can record a test in the script language of the tool while you perform it manually. By running the script, you can let the tool perform the test automatically

Potential problems

This pattern can be very useful to automate actions that are repeated very often and almost never change, for instance set-ups.

If you capture a test not only as a script but also as video clip you can get the information you need to automate it even if testers have no time to support you. Beware: If you want your automation scripts to be easily maintainable, you should avoid creating them solely using capture-replay: every small change in the SUT can force you to record the test again and you will get *BRITTLE SCRIPTS* or *TOOL-DRIVEN AUTOMATION*.

4.1.9. CELEBRATE SUCCESS (Process Pattern)

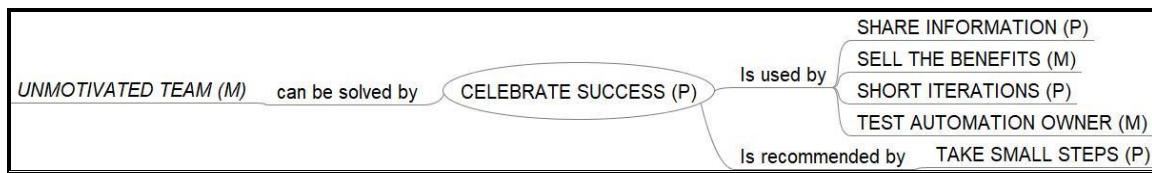


Figure 4.1.9-1 CELEBRATE SUCCESS

Pattern summary

When you have reached an important milestone, it's celebration time.

Category

Process

Context

This pattern is useful when you need support from management, testers, developers or other specialists, and where you are aiming to establish a long-lasting automation regime. It helps to keep the test automation team motivated to continue to improve the automation.

Description

When you have reached a milestone in your automation, it's time to celebrate. A milestone might be automating your first test suite, establishing a level of coverage, or a number of unattended automated tests being run.

Implementation

Invite the team, the testers and all the people that supported you (for instance that database expert that helped with some special SQL statements).

Depending on what you celebrate, you can offer:

- Special treats for coffee time
- Drinks
- Pizzas
- After work party
- Dinner at a restaurant
- Other entertainment
- Small presents
- A bonus

Celebrations don't need to be large or expensive - put some thought into what would be appreciated - sometimes just some recognition of the effort that has gone into the automation is what is needed.

Recommendations

Invite your managers to every celebration. Show what you have already accomplished and tell them that you appreciate the support they have given you already. Share your plans for the future of the automation. (This can be a subtle way to remind them that you need their continuing support for further progress).

Suggestions:

- calculate how much time has been saved by running automated instead of manual tests
- show how test coverage has grown due to test automation
- explain how your test automation solution works
- tell about an important regression defect that has been found by the automated tests before delivery of the new release

Potential problems

Remember that not everyone likes the same kind of celebration. Having a party might be fantastic for some, but others might prefer a quiet dinner or a gift voucher for books, for example.

Issues addressed by this pattern

UNMOTIVATED TEAM

4.1.10. CHAINED TESTS (Design Pattern)

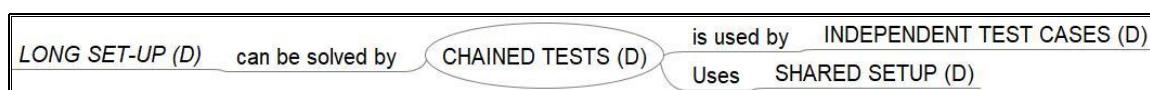


Figure 4.1.10-1 CHAINED TESTS

Pattern summary

Automate the tests so that they run in a predefined sequence.

Category

Design

Context

Use this pattern when tests need a very long setup. If possible avoid it.

Description

You automate the tests so that they run in a predefined sequence. Each test generates the initial conditions for the following ones so that you have to do the set-up only once.

Implementation

You leave the Software Under Test (SUT) as is after each test, but before you start the next test you check that the expected initial conditions are really met (the preceding tests could have failed).

Another way to avoid a long set-up for every test case is to use a SHARED SETUP that is set-up before your tests run. Each test cleans up after it is run so that the next tests can start with a clean fixture.

Potential problems

Avoid this pattern if you don't want to get the issue *INTERDEPENDENT TEST CASES*

Issues addressed by this pattern

LONG SET-UP

4.1.11. CHECK-TO-LEARN (Process Pattern)



Figure 4.1.11-1 CHECK-TO-LEARN

Pattern summary

Let new team members learn by checking the health of the existing automation testware

Category

Process

Context

Use this pattern to teach new team members what is already available to the test automation team. This pattern can be used in any context where new people have to be integrated in the automation team

Description

New team members have to learn what is already available and what are "the rules of the game". Asking them to check the health of the existing automation testware has two big advantages:

- People are positively motivated to dig in.
- The check is done from a fresh point of view.
- By comparing the existing testware with their own experience, newcomers can introduce new ideas, tools and processes.

Implementation

Ask newcomers not just to learn how the tool works, but to check the testware. Some suggestions:

- Are the testcases sufficiently documented?
- How easy is it to find scripts or data, particularly for reuse?
- How easy is it to write a new test?
- How easy is it to change a test if the software being tested has changed?
- How easy is it to find out what went wrong when an automated test fails?

They can keep track of how long it takes to do these tasks - the quicker and easier it is, the healthier the automation.

Potential problems

Because the newcomers are (obviously) new, they may get a bit lost - but this is actually very useful information about the automation!

Be sure to emphasise that what is being investigated is the automation, not the person! (They don't "fail" the test, the automation does!)

Issues addressed by this pattern

LIMITED EXPERIENCE

4.1.12. COMPARE WITH PREVIOUS VERSION (Execution Pattern)

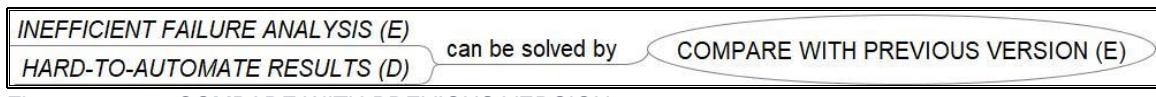


Figure 4.1.12-1 COMPARE WITH PREVIOUS VERSION

Pattern summary

Compare the current results with the results got by running the same tests in the previous version of the SUT

Category

Execution

Context

This pattern can be used when you have verified results for the previous release

Description

When it's difficult to judge if the results are OK because of so many relationships, it may be much easier to just compare the current results with the results you got by running the same tests in the previous version of the SUT. If you know them to be OK then you don't have to "understand" the contents, but just have to look for differences.

Implementation

First of all you must set-up the same initial conditions in the old and in the new version. Then run the tests on the old version and on the new one. Compare the results.

Potential problems

If you don't have automated tests for the older version, run them manually

Issues addressed by this pattern

*INEFFICIENT FAILURE ANALYSIS
HARD-TO-AUTOMATE RESULTS*

4.1.13. COMPARISON DESIGN (Design Pattern)

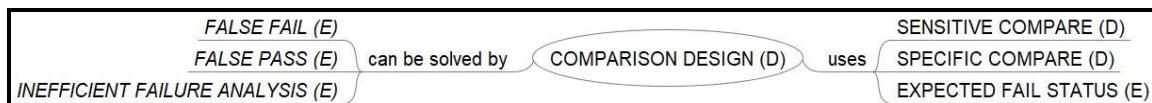


Figure 4.1.13-1 COMPARISON DESIGN

Pattern summary

Design the comparison of test results to be as efficient as possible, balancing Dynamic and Post-Execution Comparison, and using a mixture of Sensitive and Robust/ Specific comparisons.

Category

Design

Context

This pattern is applicable to any automated test.

Description

Automated comparison of test results can be done while a test is running (Dynamic Comparison) or after a test has completed (Post-Execution Comparison). Results that could influence the progress of a test should be done during the test, but the comparison of the contents of a file or database is best done after the test has completed. Choosing the right type of comparison will give more efficient automation.

Test sensitivity is related to the amount that is compared in a single comparison. A SENSITIVE COMPARE compares as much as possible, e.g. a whole screen, and a SPECIFIC COMPARE compares the minimum that is useful for the test, e.g. a single field.

Implementation

Dynamic comparison are programmed into the script of a test so that they are carried out during the execution of that test.

Post-execution comparisons are carried out as a separate step after a test has completed execution, as part of post-processing for that test, or as a separate activity.

Sensitive tests look at a large amount of information, such as an entire screen or window (possibly using masks or filters to exclude any results that this test is not looking at). See SENSITIVE COMPARE.

Specific tests look at only the specific information that is of interest to a particular test. See SPECIFIC COMPARE.

Use Sensitive tests for high level Smoke tests or Breadth tests, and use Specific tests for Depth tests or detailed tests of functions and features.

Potential problems

If you have Dynamic comparisons that would be better as Post-Execution comparison, your tests will take a lot longer to run than is necessary.

If you have Post-Execution comparison that would be better as Dynamic comparison, then you won't be able to use the intermediate results of tests to skip over irrelevant steps or abort tests that it's not worth continuing with, so you will be wasting time and the tests will run for longer than is necessary.

If all of your tests are Specific, you will miss unexpected changes which could be serious bugs, so your tests will be passing even though there are problems which should fail the tests (*FALSE PASS*).

If all of your tests are Sensitive, every unexpected problem will trip up all of your tests, even though you are not interested in this after the first time (*FALSE FAIL*). You could use EXPECTED FAIL STATUS to overcome this if you can't change enough of the tests to be Specific. The tests will also take longer to run, as there will be more checking to do for each test.

Issues addressed by this pattern

FALSE FAIL

FALSE PASS

INEFFICIENT FAILURE ANALYSIS

4.1.14. DATA-DRIVEN TESTING (Design Pattern)

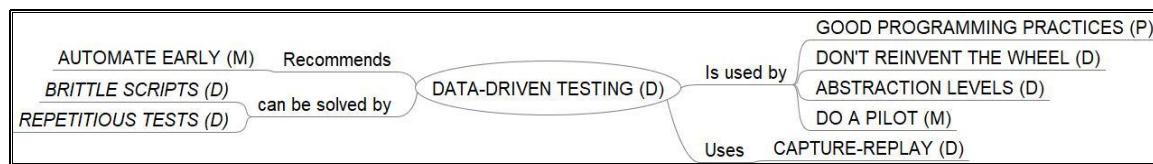


Figure 4.1.14-1 DATA-DRIVEN TESTING

Pattern summary

Write the test cases as scripts that read their data from external files.

Category

Design

Context

One of the most used patterns to easily develop modular automation scripts for long lasting automation.

Description

Write the test cases as scripts that read their data from external files. In this way you have only one script to drive the tests but by changing the data you can create any number of test cases. The charm is that if you have to update the script because of some change in the Software Under Test (SUT), you frequently don't have to change your data, so you don't have to spend too much effort in maintenance.

Implementation

You write a script with variables whose content is read sequentially from a file such as a spreadsheet. Every line in the file delivers the data for a different test case. An easy way to implement this pattern is to use CAPTURE-REPLAY to capture the tests initially. The captured test will have constant data (i.e. specific test inputs for every field). You can then replace these constants with a call to data that is read from an external file.

Potential problems

If your data is not contained in only one data file, you must make sure that the script and data are correctly matched.

Issues addressed by this pattern

BRITTLE SCRIPTS
REPETITIOUS TESTS

4.1.15. DATE INDEPENDENCE (Design Pattern)

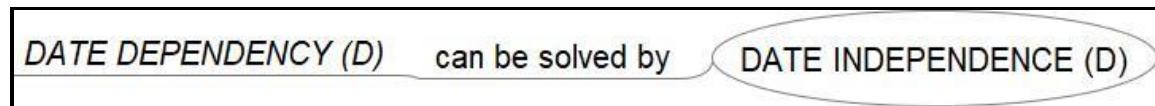


Figure 4.1.15-1 DATE INDEPENDENCE

Pattern summary

Write your test cases to be date independent

Category

Design

Context

Use this pattern if your test cases or your expected results have some dependency on dates.

Description

If possible redesign your test cases (or expected results) to avoid DATE DEPENDENCY. If this is not possible, you must set the initial conditions accordingly

Implementation

Here some suggestions on how to implement this pattern:

- Ask developers to use the current date as default in the date fields, then you don't have to use it in your scripts.
- Calculate the dates that you need by adding or subtracting the wanted difference from the current date.

- In the setup procedure, set the system date to the date you are expecting (after you have saved the current date); after the test has been run set the current date back.
- In the expected results use only date & time differences and not the actual dates.
- Ask developers to build a switch that you can use to block showing date & time on outputs.
- Use for comparisons a tool that allows you to blot out the variable areas (date & time) on your outputs.

Potential problems

If you have changed the system date and for some reason a test doesn't run through, your machine will be running on a wrong date and you may not notice it immediately

Issues addressed by this pattern

DATE DEPENDENCY

4.1.16. DEDICATED RESOURCES (Management Pattern)



Figure 4.1.16-1 DEDICATED RESOURCES

Pattern summary

The test automation team is available full time and all the needed tools and machines are in place.

Category

Management

Context

This pattern is applicable if you have a tight schedule for your test automation project, or if you want to build a significant test automation regime that will be used by many testers.

This pattern is not applicable for very small organisations where, for example, there are only one or two testers, and they are doing automation work in parallel with testing activities.

Description

A test automation team is assigned to work on test automation full-time, without other responsibilities. Any, tools, utilities, machines or other resources are provided as needed. You can get the help of specialists when you need it.

Implementation

Some suggestions:

- A test automation project requires possibly more effort than a software development project and should be done as a FULL TIME JOB.
- PLAN SUPPORT ACTIVITIES so that you can get hold of specialists (for instance data base managers) when you need them.
- Tools and machines to run the tests need to be available, and ideally dedicated to running automated tests. GET ON THE CLOUD for cost-efficiency.
- SHARE INFORMATION: tell management that you will not be able to finish the project successfully if the team cannot concentrate completely on it. If this is not possible for the duration of the project, then at least at the beginning of the project.
- If you have *FLAKY TESTS* you should check if the cause could be that your resources (machines, databases, network) are used concurrently by some other application. If this is the case, see to it that you get the resources you need when you need them for your exclusive use.

Potential problems

It is often the case that resources are severely limited but expecting people to "do test automation in their spare time" is very unlikely to work and is not efficient.

Issues addressed by this pattern

AD-HOC AUTOMATION

FALSE FAIL

FLAKY TESTS

INADEQUATE TECHNICAL RESOURCES

INCONSISTENT DATA

NO PREVIOUS TEST AUTOMATION

SCHEDULE SLIP

HARD-TO-AUTOMATE RESULTS

4.1.17. DEFAULT DATA (Design Pattern)

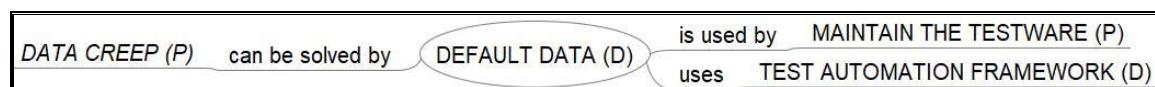


Figure 4.1.17-1 DEFAULT DATA

Pattern summary

Use default data to simplify data input

Category

Design

Context

This pattern is useful when test cases need a lot of common data that is not all meaningful for the specific test.

Description

Instead of cluttering test cases with data that is needed, but is not relevant for the specific test, use default data.

For example, if you are interested in testing validation for a Name field but don't care about the contents of the address or phone number fields, set up a default value for the address ("123 Main Street") and phone number. Then in your test, you will say what the Name field should contain, but not what the other fields should contain, so your framework will copy the default values (because if they were left blank, the test would fail) to those fields.

Implementation

Prepare default data. Your TEST AUTOMATION FRAMEWORK substitutes default data for missing data during the data inputs in the test cases.

Potential problems

Default values should be valid, but there may be combinations of other values that make a normally valid value invalid.

You may end up with a lot of people living at the same address... ;-)

Issues addressed by this pattern

DATA CREEP

4.1.18. DEPUTY (Management Pattern)

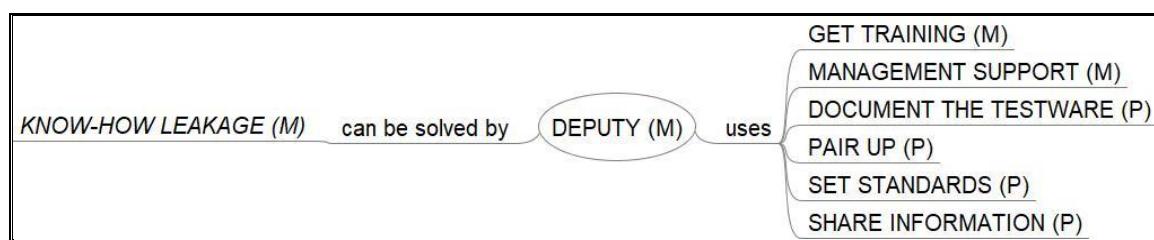


Figure 4.1.18-1 DEPUTY

Pattern summary

Appoint a deputy to all important knowledge carriers

Category

Management

Context

This pattern is important for long lasting automation, it's not needed for one-off or disposable scripts.

Description

It is common knowledge that one needs to back up the testware (data or scripts) just in case it could be lost or damaged. The same should also happen for the know-how in the heads of the automation engineers: there should always be a deputy that is able to take over if who is in charge of the framework, the scripts or test execution suddenly gets sick, goes on vacation or, in the worst case, leaves the company.

Implementation

Here some suggestions:

- GET TRAINING: keep all members of the automation team up to date with the latest tools and standards
- DOCUMENT THE TESTWARE: This pattern is essential for keeping the test automation effort maintainable.
- PAIR UP, SHARE INFORMATION: apply these patterns to spread knowledge though the team.
- SET STANDARDS: it's easier to learn and understand what other people do, if everybody follows the same standards.
- MANAGEMENT SUPPORT: you will need it in order to free people to become deputies.

Potential problems

Beware of people building little "empires", that don't want to share their knowledge with the rest of the team.

Issues addressed by this pattern

KNOW-HOW LEAKAGE

4.1.19. DESIGN FOR REUSE (Design Pattern)

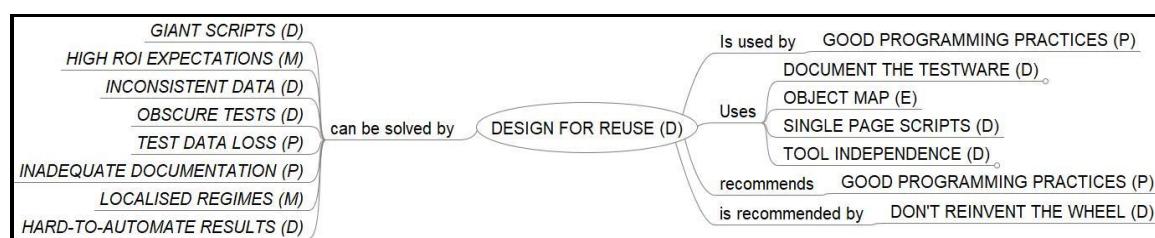


Figure 4.1.19-1 DESIGN FOR REUSE

Pattern summary

Design reusable testware.

Category

Design

Context

Use this pattern for long lasting and maintainable automation. It isn't necessary for disposable scripts.

Description

Design modular scripts: write script code for some functionality in only one place and call it when needed. In this way you can reuse the components again and again and if you do need to change something you have to implement and test it only in one place.

Reuse test data as much as possible.

Implementation

- When you see that you need to use again a part of some script (keyword), extract it and build with it a new script (keyword) that you can then call from the original script and from any future scripts (keywords) that may need it. Be sure to document exactly what it does and what parameters must be passed and give it a name that makes it easy to find.
- SINGLE PAGE SCRIPTS: write a script for every screen you want to automate, where you include all the GUI-objects on the screen. The input parameters can be used to communicate which GUI-objects will be activated in the specific test case and how they will be activated. For example, if a parameter is missing, the respective GUI-object will be ignored.
- Implement an OBJECT MAP: if the objects get meaningful names it will be easier for the testers to understand the scripts. Also, in this way you pave the way for your scripts to achieve TOOL INDEPENDENCE.
- Write a script for all the various set-ups that you will need. Again, define with input parameters the specifics for each test case.
- Generate test data that can be reused repeatedly (for instance a customer record with some specific characteristics).
- DOCUMENT THE TESTWARE: give the “building blocks” (scripts or data) descriptive names so that a tester can see at a glance what is available, how to use it and when to use it.

Recommendations

If possible, get a developer to coach you in GOOD PROGRAMMING PRACTICES.

Potential problems

If your developers don't implement GOOD PROGRAMMING PRACTICES then try to learn from books, conferences etc.

Issues addressed by this pattern

GIANT SCRIPTS

HIGH ROI EXPECTATIONS

*INADEQUATE DOCUMENTATION
INCONSISTENT DATA
LOCALISED REGIMES
OBSCURE TESTS
TEST DATA LOSS
HARD-TO-AUTOMATE RESULTS*

4.1.20. DO A PILOT (Management Pattern)

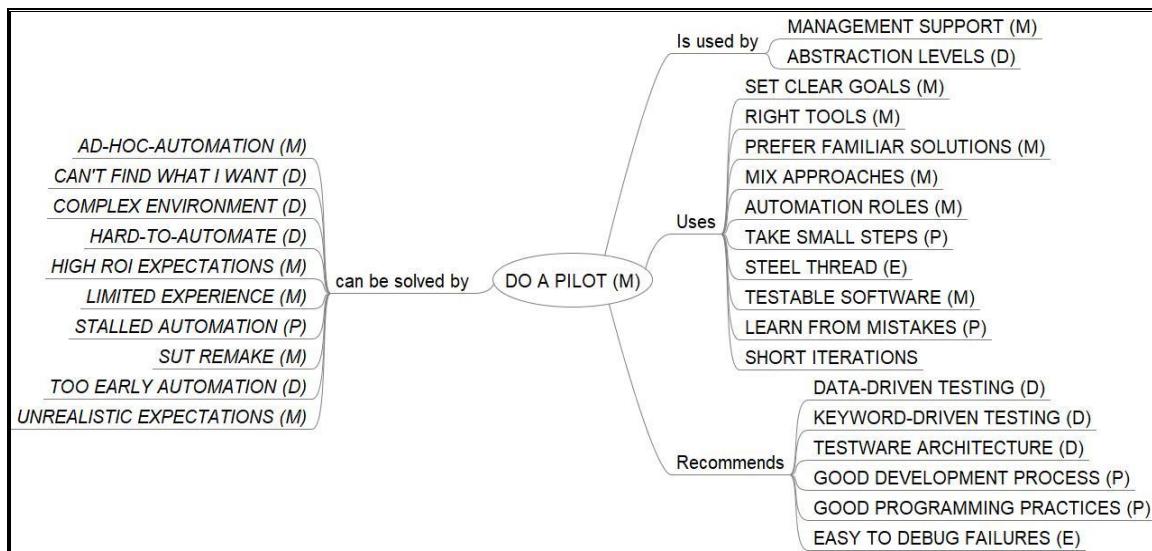


Figure 4.1.20-1 DO A PILOT

Pattern summary

Start a pilot project to explore how to best automate tests on the application.

Category

Management

Context

This pattern is useful when you start an automation project from scratch, but it can also be very useful when trying to find the reasons your automation effort is not as successful as you expected.

This pattern is not needed for one-off or disposable scripts.

Description

You start a pilot project to explore how to best automate tests on your application. The advantage of such a pilot is that it is time boxed and limited in scope, so that you can concentrate in finding out what the problems are and how to solve them. In a pilot project nobody will expect that you automate a lot of tests, but that you find out what are the best tools for your application, the best design strategy and so on.

You can also deal with problems that occur and will affect everyone doing automation and solve them in a standard way before rolling out automation

practices more widely. You will gain confidence in your approach to automation. Alternatively, you may discover that something doesn't work as well as you thought, so you find a better way - this is good to do as early as possible! Tom Gilb says: "If you are going to have a disaster, have it on a small scale"!

Implementation

Here some suggestions and additional patterns to help:

- First of all, SET CLEAR GOALS: with the pilot project you should achieve one or more of the following goals:
 - Prove that automation works on your application.
 - Choose a test automation architecture.
 - Select one or more tools.
 - Define a set of standards.
 - Show that test automation delivers a good return on investment.
 - Show what test automation can deliver and what it cannot deliver.
 - Get experience with the application and the tools.
- Try out different tools in order to select the RIGHT TOOLS that fit best for your SUT, but if possible PREFER FAMILIAR SOLUTIONS because you will be able to benefit from available know-how from the very beginning.
- Do not be afraid to MIX APPROACHES.
- AUTOMATION ROLES: see that you get the people with the necessary skills right from the beginning.
- TAKE SMALL STEPS, for instance start by automating a STEEL THREAD: in this way you can get a good feeling about what kind of problems you will be facing, for instance check if you have TESTABLE SOFTWARE.
- Take time for debriefing when you are thru and don't forget to LEARN FROM MISTAKES.
- In order to get fast feedback, adopt SHORT ITERATIONS.

What kind of areas are explored in a pilot? This is the ideal opportunity to try out different ways of doing things, to determine what works best for you. These three areas are very important:

- Building new automated tests. Try different ways to build tests, using different scripting techniques (DATA-DRIVEN TESTING, KEYWORD-DRIVEN TESTING). Experiment with different ways of organising the tests, i.e. different types of TESTWARE ARCHITECTURE. Find out how to most efficiently interface from your structure and architecture to the tool you are using. Take 10 or 20 stable tests and automate them in different ways, keeping track of the effort needed.
- Maintenance of automated tests. When the application changes, the automated tests will be affected. How easy will it be to cope with those changes? If your automation is not well structured, with a good TESTWARE ARCHITECTURE, then even minor changes in the application can result in a disproportionate amount of maintenance to the automated tests - this is what often "kills" an automation effort! It is important in the pilot to

experiment with different ways to build the tests in order to minimise later maintenance. Putting into practice GOOD PROGRAMMING PRACTICES and a GOOD DEVELOPMENT PROCESS are key to success. In the pilot, use different versions of the application - build the tests for one version, and then run them on a different version, and measure how much effort it takes to update the tests. Plan your automation to cope the best with application changes that are most likely to occur.

- Failure analysis. When tests fail, they need to be analysed, and this requires human effort. In the pilot, experiment with how the failure information will be made available for the people who need to figure out what happened. What you want to have are EASY TO DEBUG FAILURES. A very important area to address here is how the automation will cope with common problems that may affect many tests. This would be a good time to put in place standard error-handling that every test can call on.

Potential problems

Don't bite off more than you can chew: if you have too many goals you will have problems achieving them all.

Do the pilot on something that is worth automating, but not on the critical path. Make sure that the people involved in the pilot are available when needed - managers need to understand that this is "real work"!

Issues addressed by this pattern

AD-HOC AUTOMATION

CAN'T FIND WHAT I WANT

COMPLEX ENVIRONMENT

HARD-TO-AUTOMATE

HIGH ROI EXPECTATIONS

LIMITED EXPERIENCE

STALLED AUTOMATION

SUT REMAKE

TOO EARLY AUTOMATION

UNREALISTIC EXPECTATIONS

4.1.21. DOCUMENT THE TESTWARE (Process Pattern)

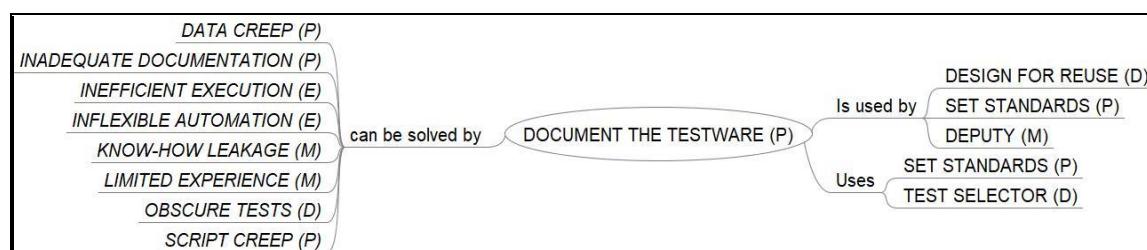


Figure 4.1.21-1 DOCUMENT THE TESTWARE

Pattern summary

Document the automation scripts and the test data.

Category

Process

Context

This pattern is essential for long lasting and maintainable automation. It's not necessary for disposable scripts.

Description

Document the automation scripts and the test data, so that they are:

- easily found
- understandable
- reusable
- traceable (e.g. to requirements)

Implementation

Some suggestions:

- Documentation will be easier to write, maintain and read if you SET STANDARDS.
- Descriptive names go a long way in documenting what some testware is or does.
- use naming conventions consistently to make available testware easy to find.
- Use a standard template as the Test Description: document in every script or batch file:
 - What does it do.
 - How do you call it.
 - What does it return.
 - other relevant test characteristics (e.g. a smoke test, performance test, feature tested, TEST SELECTOR tags).
- Put the documentation in configuration management together with the testware and the corresponding release of the SUT.
- If your current documentation is incomplete, let newbies update it: they learn faster and you get a better documentation.

Recommendations

Automating the tests is also a good time to extract expert knowledge from your manual testers and to document it as automated test cases

Issues addressed by this pattern

DATA CREEP

INADEQUATE DOCUMENTATION

INEFFICIENT EXECUTION

INFLEXIBLE AUTOMATION

*KNOW-HOW LEAKAGE
LIMITED EXPERIENCE
OBSCURE TESTS
SCRIPT CREEP*

4.1.22. DOMAIN-DRIVEN TESTING (Design Pattern)



Figure 4.1.22-1 DOMAIN-DRIVEN TESTING

Pattern summary

Develop a domain-specific language for testers to use when writing their automated test cases.

Category

Design

Context

This pattern is appropriate when testers should be able to write and run automated tests even if they are not adept with the automation tools, or if you want them to start writing test cases for automation before the Software Under Test (SUT) has been completely developed.

The pattern is not appropriate for very small-scale or one-off automation efforts.

Description

Testers develop a simple domain-specific language to write their automated test cases with. Practically this means that actions particular to the domain are described by appropriate commands, each with a number of required parameters. As example let's imagine that we want to insert a new customer into our system. The domain-command will look something like this:

`New_Customer (FirstName, LastName, HouseNo, Street, ZipCode, City, State)`

Now testers only have to call `New_Customer` and provide the relevant data for a customer to be inserted. Once the language has been specified, testers can start writing test cases even before the SUT has actually been implemented.

Implementation

To implement a domain-specific language, scripts or libraries must be written for all the desired domain-commands. This is usually done with a TEST AUTOMATION FRAMEWORK that supports ABSTRACTION LEVELS.

There are both advantages and disadvantages to this solution. The greatest advantage is that testers who are not very adept with the tools can write and maintain automated test cases. The downside is that you need developers or test automation engineers to implement the commands so that testers are completely dependent on their “good will”. Another negative point is that the domain libraries may be implemented in the script language of the tool, so that to change the tool may mean to have to start again from scratch (*TOOL DEPENDENCY*). This can be mitigated to some extent using ABSTRACTION LEVELS.

KEYWORD-DRIVEN TESTING is a good choice for implementing a domain-specific language: Keyword = Domain-Command.

Potential problems

It does take time and effort to develop a good domain-driven automated testing infrastructure.

Issues addressed by this pattern

LATE TEST CASE DESIGN

MANUAL MIMICRY

NON-TECHNICAL-TESTERS

OBSCURE TESTS

TOO EARLY AUTOMATION

4.1.23. DON'T REINVENT THE WHEEL (Design Pattern)

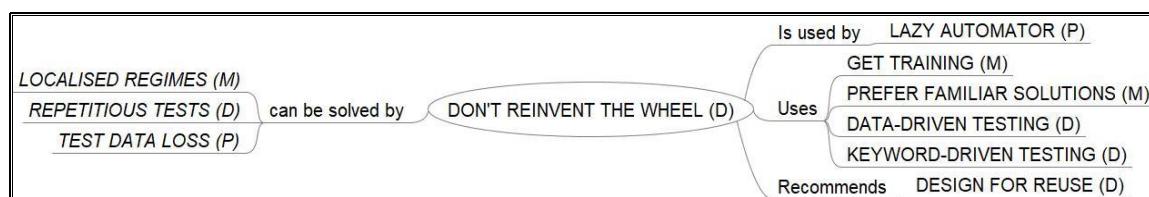


Figure 4.1.23-1 DON'T REINVENT THE WHEEL

Pattern summary

Use available know-how, tools and processes whenever possible.

Category

Design

Context

Use this pattern for efficient automation, but it is always useful!

Description

Use available know-how, tools and processes whenever possible. Reuse available data. Reuse scripts or libraries: you save lots of effort because you can just use testware that has already been tested and implemented.

James Tony: Duplication is to be avoided! Copy and paste is (usually) the wrong thing to do – repeated cloning and tweaking of tests can lead to high technical debt and having to make the same change in many places when the requirements change

Implementation

Some suggestions:

- GET TRAINING: you don't have to invent everything yourself!
- PREFER FAMILIAR SOLUTIONS: take advantage of what know-how is already available in your company.
- For *REPETITIOUS TESTS* implement DATA-DRIVEN TESTING.
- Look for common keywords and implement KEYWORD-DRIVEN TESTING.

Recommendations

DESIGN FOR REUSE: Write scripts so that they can be reused and reuse them. Create generalized data that can be reused and reuse it

Issues addressed by this pattern

LOCALISED REGIMES

REPETITIOUS TESTS

TEST DATA LOSS

4.1.24. EASY TO DEBUG FAILURES (Execution Pattern)

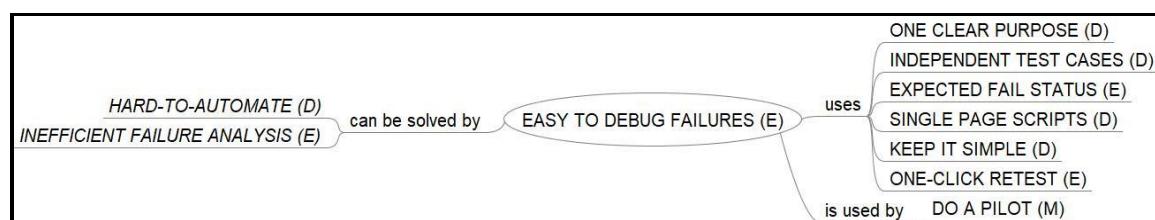


Figure 4.1.24-1 EASY TO DEBUG FAILURES

Pattern summary

Each test failure should make it obvious which test failed and what went wrong. It shouldn't take hours to figure this out and it shouldn't require a rerun of the test.

Category

Execution

Context

This pattern is useful for big and small automation undertakings.

Description

Develop simple clear well documented tests. Design the automated test so that it keeps track of the information you would want to know when the test fails, for example, capture screen shots on a regular basis, or log all messages that are

output, or record all the input data in one place. Automate the clear up such that all the test results and associated information is available in one place and is easily accessible to the person investigating the reason for the failure.

Implementation How to implement this pattern depends on your context. Here are some suggestions:

- Develop tests with ONE CLEAR PURPOSE: if such a test fails it will be easier to understand why.
- Implement INDEPENDENT TEST CASES: in this way you can be sure that the failure is not caused by an earlier test.
- If many tests are failing for the same (minor) bug, implement EXPECTED FAIL STATUS.
- If you write SINGLE PAGE SCRIPTS you will know at a glance in which page or window the failure occurred.
- And last but not least: KEEP IT SIMPLE!

Potential problems

If you do need to rerun your tests for better checking, implement ONE-CLICK RETEST.

Issues addressed by this pattern

HARD-TO-AUTOMATE

INEFFICIENT FAILURE ANALYSIS

4.1.25. EXPECT INCIDENTS (Execution Pattern)



Figure 4.1.25-1 EXPECT INCIDENTS

Pattern summary

Automated scripts should be able to react to unexpected incidents without disrupting execution

Category

Execution

Context

This pattern is especially useful for long living automation where unexpected changes would otherwise disrupt execution.

Description

Incidents can be:

- Unexpected windows or pop-ups.
- Windows that sometimes show up and sometimes don't.

- Actions that take variable time to execute.

Your test scripts must be able to react accordingly so that execution can continue.

Implementation

Depending on the type of incidents, here are some suggestions about how to react:

- Unexpected Windows or pop-ups: modern tools offer the option to just click them off.
- For windows that don't always show up or actions that take variable time to execute: look up the pattern VARIABLE DELAYS.

Potential problems

Be careful when clicking off an unexpected window: it could be a bug! Instead of ignoring it, write a warning in the log.

Issues addressed by this pattern

INEFFICIENT EXECUTION

4.1.26. EXPECTED FAIL STATUS (Execution Pattern)



Figure 4.1.26-1 EXPECTED FAIL STATUS

Pattern summary

Comparing to an Expected Fail outcome rather than the Expected Result enables automated failure analysis when there is a minor bug affecting all tests.

Category

Execution

Context

This pattern is useful when there is a minor bug that isn't going to be fixed anytime soon, which is affecting a large number of automated tests. Every test is failing because of the same unimportant bug. But every failing automated test has to be investigated - this makes the failure analysis time become excessive, which is not good - wasteful. Sometimes people just don't bother to run the tests - then you have lost the value of the automated tests. Setting up an additional Test Status (not just Pass or Fail) gives additional benefit from automation and overcomes the problem.

Description

In addition to the "normal" test status of "Pass" or "Fail", two additional test statuses are needed (and even more can also be useful).

- Pass: the test actual results match the Expected Results.

- Fail: the test actual results do not match the Expected Results.
- Expected Fail: the test results match the "Expected Fail Results" which include the minor bug as part of this test result.
- Unknown: the test results do not match the Expected Fail Results.

Implementation

Using more than two test statuses needs to be set up ideally from the beginning of an automation effort, so that every test has the option of using additional statuses. You need to specify which tests are to be compared to Expected Fail Results, and the Expected Fail Results need to be set up for each of the tests that will use it. This may just be an (automated) update to a set of Expected Results.

When a test is run that does not match the Expected Fail Result, there are two possibilities - it may be that this minor bug has now been fixed, so running the test immediately against the Expected Result would result in a Pass. Or it could be that there is a new bug that the automated test has now found in addition to the known bug - in this case the test status is Unknown and needs to be looked at.

Expected Fail is not the only useful additional test status - you could have a test status for any of the following (or others):

- Tests blocked due to environment problem (e.g. network down, timeouts).
- Set-up problem (e.g. files missing).
- Test needs to be changed but hasn't been updated yet.

Potential problems

It does take time to set up the facility to use more than two test statuses, but the benefits are also significant.

Issues addressed by this pattern

INEFFICIENT FAILURE ANALYSIS

4.1.27. FAIL GRACEFULLY (Execution Pattern)

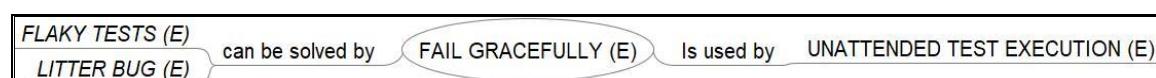


Figure 4.1.27-1 FAIL GRACEFULLY

Pattern summary

If a test fails it should restore the system and the environment so that the successive tests are not affected.

Category

Execution

Context

This pattern is applicable if you want your test automation to run unattended, and where the system being tested is mature and you are not expecting a lot of failures.

This pattern is not appropriate if you will need to do failure analysis after a test has completed (or failed), or for one-off scripts.

Description

See to it that when a test fails, you clean-up and exit, so that the next tests can be performed normally.

Implementation

Build in your scripts error-catching functionality that resets the system and the environment and exits the failed test.

Potential problems

This pattern is the opposite approach to FRESH SETUP, where the tests don't clean up after they have been run.

Issues addressed by this pattern

FLAKY TESTS

LITTER BUG

4.1.28. FRESH SETUP (Design Pattern)



Figure 4.1.28-1 FRESH SETUP

Pattern summary

Before executing each test prepares its initial conditions from scratch. Tests don't clean up afterwards.

Category

Design

Context

Use this pattern for long lasting and maintainable automation, but it can be useful also when writing disposable scripts.

Description

Each test prepares its initial conditions from scratch before executing, so that it makes sure to run under defined initial conditions. In this way each test can run independently from all other tests. Tests don't clean up afterwards, so that if you want to check the results you can immediately control the state of the Software Under Test (SUT).

Implementation

Initial conditions can be very diverse. Here some suggestions:

- Database configuration:
 1. Copy the table structure of the database for the current release of the SUT: in this way you will always have the current database. Since this may take some time you should do it only once at the beginning of a test suite.
 2. Insert in the database the standard configuration data that you will need for each of the following test cases. This should also be done only once at the beginning of the test suite.
 3. For each test case: make sure the relevant variable database entries are empty. If not, remove or initialise content (after ensuring that you have not wiped out the traces of prior test errors).
 4. For each test case: insert the variable data that your test case expects.
- File configuration:
 - Copy input or comparison files to a predefined standard directory.
- SUT:
 - to be sure that the SUT is in the required state, it should be started anew for each test case and driven to the foreseen starting point.
- Virtual machines:
 - for complex environments it may pay to start each time from a known VM snapshot. This has the added benefit that in the case of an erratic test then the failed vm can be automatically stored away for further analysis. Storage and time limitations would probably make this impractical for non-regression scenarios where you expect a lot of failures.

Leave the SUT as it is after the test is run. In this way, after running the test, you can immediately check the state of the SUT, database contents etc. without having to restart the test and stopping it before it cleans up. Even if you do have to restart it, e.g. because it was followed by other tests, you don't have to change the scripts in any way to repeat the test and check the results.

Potential problems

If the setup is very slow, you should consider using instead a SHARED SETUP

Issues addressed by this pattern

FALSE FAIL

INCONSISTENT DATA

INEFFICIENT FAILURE ANALYSIS

INTERDEPENDENT TEST CASES

HARD-TO-AUTOMATE RESULTS

4.1.29. FULL TIME JOB (Process Pattern)



Figure 4.1.29-1 FULL TIME JOB

Pattern summary

Arrange for the test automation team to be available full time.

Category

Process

Context

This pattern is appropriate for larger organisations, or for those who are establishing a solid starting point for long lasting, maintainable test automation. It is a good idea to have a pilot automation project staffed by a few people full-time for a few months.

Once your automation is well-established, you may not need automators full-time, so this pattern may be a temporary one for you.

This pattern is not appropriate for small organisations, or where one person must be both tester and automator, or when there are severe resource restrictions.

Description

Test automation is quite a complex task, so it is very helpful if team members are dedicated to the task and are able to concentrate full time on test automation. The full-time team may be formed for a relatively short time, e.g. 3 to 6 months, or it could be for a longer time (several years). In larger organisations, there may be a permanent full-time team dedicated to test automation.

Implementation

With MANAGEMENT SUPPORT you should be able to free your team members from other tasks.

Working full time on the test automation team will also be a boost to motivation, because by concentrating on one job, people can work better and so get more satisfaction from what they're doing.

Recommendations

Inform management of the benefits of having the team work exclusively on the test automation project.

SHARE INFORMATION to management about the risks of not being able to implement automation as successfully if the team cannot concentrate completely on it, at least for a time.

Potential Problems

If you do most of the automation work on the side-line, maybe because you found it really interesting, you let management believe that everybody will be doing it just like you and they will expect the same from everybody. This is a good way to lose good people if they are not willing to sacrifice all their free time for the job.

Issues addressed by this pattern

INADEQUATE TEAM

SCHEDULE SLIP

UNMOTIVATED TEAM

4.1.30. GET ON THE CLOUD (Management Pattern)



Figure 4.1.30-1 GET ON THE CLOUD

Pattern Summary

Instead of using local machines put the project on one or more virtual machines

Category

Management

Context

Use this pattern if you have a distributed team or if you are supposed to run the same test cases in different environments

Description

Instead of using local machines put the project on one or more virtual machines. With virtual machines you can also set-up as many test environments as you need

Implementation

The virtual machines can be in your intranet or at some provider. Be sure that all team members have the necessary authorizations and that all the necessary tools or environments are installed

Issues addressed by this pattern

HARD-TO-AUTOMATE

INADEQUATE COMMUNICATION

INADEQUATE TECHNICAL RESOURCES

4.1.31. GET TRAINING (Management Pattern)



Figure 4.1.31-1 GET TRAINING

Pattern summary

Plan to get training for all those involved in the test automation project.

Category

Management

Context

This pattern is especially useful when you are starting a new automation project or when you have to integrate new people on the team, but actually you may need it also in order to write disposable scripts.

Description

All those involved in the test automation project should get some kind of training. Depending on their assignment they will of course need different kinds or degrees of schooling.

Implementation

Some suggestions:

- Learn from colleagues that are already doing test automation (PAIR UP).
- Learn by checking the health of the existing automation testware (CHECK-TO-LEARN).
- Read test automation books or magazines: there are a lot of useful books on the market, that not only give tips on what to do, but also on what not to do.
- Go to test conferences: there are always very informative test automation tracks.
- Subscribe to webinars: quite often they present interesting issues and they are mostly free.
- Tool vendors offer courses and training on how to use their tools.
- Get a coach to start you off or to help you implement a better automation strategy.
- Search the web: there are a lot of very helpful sites on test automation.

Recommendations

To earn support from your managers, you may have to explain to them why you need training. Prepare a report to show how much time you would save if you could start "as an expert" instead of learning on the job.

If you get coaching, remember that you don't need a coach all the time: After a few days you should work alone for a week or so. When the coach returns you will have gained more understanding by trying to go on your own and you will have more and better questions to ask.

Team members may be feeling unmotivated because they don't feel up to their tasks. Make sure that they can ASK FOR HELP and let them get the training they need.

Remember that there is a need for both formal training (courses or conference tutorials on specific tools or test automation principles), and also for informal training within your company. This would be useful to help new automators understand the application that is being tested, and to understand how automation has been done in the past in the company. Informal training may just be a group of people getting together to discuss a topic, or it could be an internal presentation.

Potential problems

Make sure that people sent to trainings share their new gained knowledge with the other team members. Also, don't send always only the same people to trainings.

Issues addressed by this pattern

KNOW-HOW LEAKAGE

LIMITED EXPERIENCE

LOCALISED REGIMES

NO PREVIOUS TEST AUTOMATION

UNMOTIVATED TEAM

4.1.32. GOOD DEVELOPMENT PROCESS (Process Pattern)

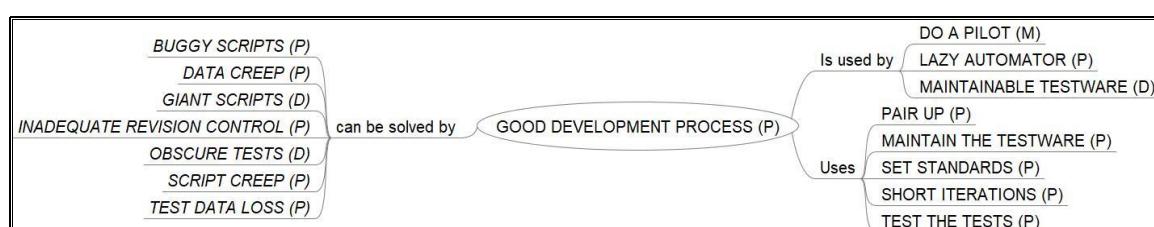


Figure 4.1.32-1 GOOD DEVELOPMENT PROCESS

Pattern summary

Use a good development process for test code; the same as developers [should] use for production code.

Category

Process

Context

This pattern is necessary if you want your automation to be long lasting. If you are just doing short term automation you will not really need it (even though it would still be helpful).

Description

Test automation is development, so you should adopt a process similar to the software development process (assuming it is a good development process). Development of test code works very well if it is an agile process.

See 'www.satisfice.com/presentations/agileauto.pdf' for a good presentation about agile automation development.

Implementation

Learn about software development principles, if you are going to be working directly with the scripting tools. There may be courses on programming from your local university, for example. Test code needs to be well structured, with small self-contained modules that call other reusable modules. Changes to the software should impact the smallest number of test modules. Scripts and all other testware should be under version control (Configuration management).

Agile development in general uses the following elements: (which work very well when developing automation)

- SHORT ITERATIONS: you don't plan everything up front, but break up the work-load into short iterations. At the beginning of the iteration you plan to do only what you can completely finish. At the end of the iteration you have working software that you could deploy to production. The customer can check if you delivered what he or she had in mind. If not, you adapt it as required in the following iterations. In the case of test automation, your code will be scripts and your customers will be the testers, so it works the same way.
- Everyone on the team is responsible for the quality of the developed software. For test automation this means that team members:
 - PAIR UP
 - TEST THE TESTS
 - MAINTAIN THE TESTWARE
 - SET STANDARDS

You can document your test automation process in a Wiki.

Potential problems

If the software developers in your organization do not have good software development practices, then your development practices will have to be better than theirs!

It might be difficult to develop the test code in an agile way, if the software development is not agile, but it will still have advantages.

Issues addressed by this pattern

BUGGY SCRIPTS

*DATA CREEP
GIANT SCRIPTS
INADEQUATE REVISION CONTROL
OBSCURE TESTS
SCRIPT CREEP
TEST DATA LOSS*

4.1.33. GOOD PROGRAMMING PRACTICES (Process Pattern)

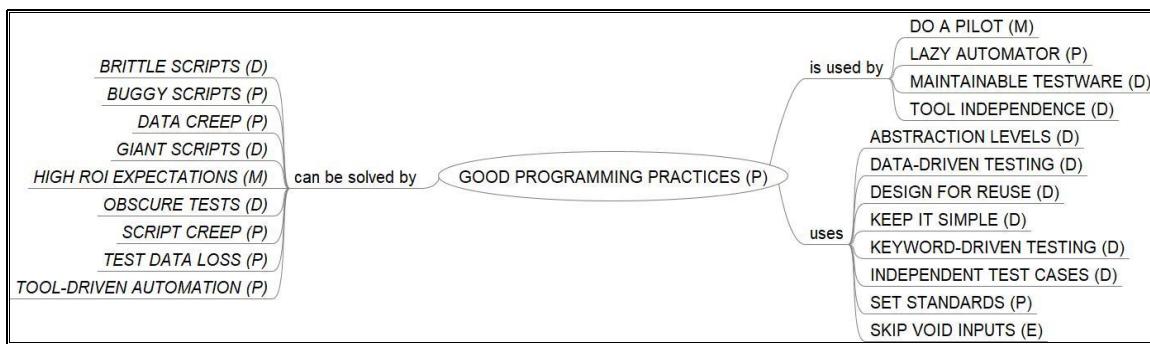


Figure 4.1.33-1 GOOD PROGRAMMING PRACTICES

Pattern summary

Use the same good programming practices for test code as in software development for production code.

Category

Process

Context

This pattern is appropriate when you want your automation scripts to be reusable and maintainable, that is when your test automation is to be long lived.

This pattern is not necessary if you only write short disposable scripts.

Description

Scripting is a kind of programming, so you should use the same good practices as in software development.

Implementation

If you don't have an automation engineer, have developers coach the automation testers. Important good practices are:

- DESIGN FOR REUSE
- KEEP IT SIMPLE
- SET STANDARDS
- SKIP VOID INPUTS
- INDEPENDENT TEST CASES

- ABSTRACTION LEVELS: Build testware that has one or more abstraction layers.
- Separate the scripts from the data: use at least DATA-DRIVEN TESTING or, better, KEYWORD-DRIVEN TESTING.
- Apply the "DRY" Principle (Don't Repeat Yourself), also known as "DIE" (Duplication is Evil).

Put your best practices in a Wiki so that both testers and developers profit from them.

Potential problems

If the software developers don't use good programming practices, then don't follow them, but investigate good programming outside of your company. (Read books!).

Issues addressed by this pattern

BRITTLE SCRIPTS

BUGGY SCRIPTS

DATA CREEP

GIANT SCRIPTS

HIGH ROI EXPECTATIONS

OBSCURE TESTS

SCRIPT CREEP

TEST DATA LOSS

TOOL-DRIVEN AUTOMATION

4.1.34. INDEPENDENT TEST CASES (Design Pattern)

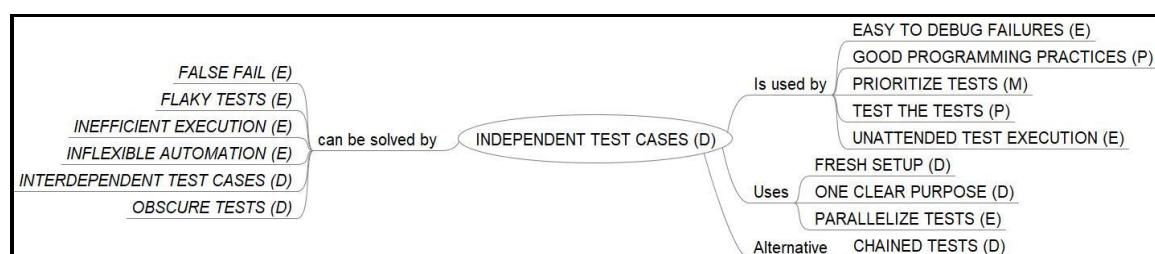


Figure 4.1.34-1 INDEPENDENT TEST CASES

Pattern summary

Make each automated test case self-contained.

Category

Design

Context

This pattern is necessary if you want to implement long lasting and efficient test automation.

An exception is when one test specifically checks the results from the prior test (e.g., using a separate test to ensure data was written into a database and not just retrieved from working memory).

It is not necessary for just writing disposable scripts.

Description

Automated test cases run independently of each other, so that they can be started separately and are not affected if tests running earlier have failed. The test may consist of a large number of different scripts or actions, some to set up the conditions needed for a test, others to execute the test steps.

Automated tests should be short and well-defined. For example, if you have one long test that takes 30 minutes to run, maybe it fails after 5 minutes the first time, after 10 the next and after 15 the 3rd time. So far you have used half an hour and have 3 bugs (which you have fixed). But you are only half-way through the test. If you separate that test into 10 tests of 3 minutes each, you can start all of them. If you can run them in parallel, execution will only take 3 minutes. Maybe 3 or 4 of them fail, but you now know that all of the others have passed, so after you fix these, the whole set of tests should be passing, and it takes a lot less time.

Implementation

- Every test starts with a FRESH SETUP before performing any action. Each test has proprietary access to its resources.
- If a test fails (stops before completion), it must reset the Software Under Test (SUT) and or the tool so that the following tests can run normally (see the pattern FAIL GRACEFULLY).
- A self-contained test does NOT mean that just one test case tests the whole application! On the contrary each test should have ONE CLEAR PURPOSE derived from one business rule.
- If you PRIORITIZE TESTS, you will also be able to run or rerun the tests independently from each other.

Potential problems

Manual tests are often performed sequentially in order to spare set-up time. They should be redesigned before automating to make sure that the automation is as efficient as possible.

If the initial set-up is very complicated or takes too much time and there is no other option, then this pattern should not be used, instead use CHAINED TESTS.

Issues addressed by this pattern

FALSE FAIL

FLAKY TESTS

INEFFICIENT EXECUTION

INFLEXIBLE AUTOMATION

INTERDEPENDENT TEST CASES

OBSCURE TESTS

4.1.35. KEEP IT SIMPLE (Design Pattern)

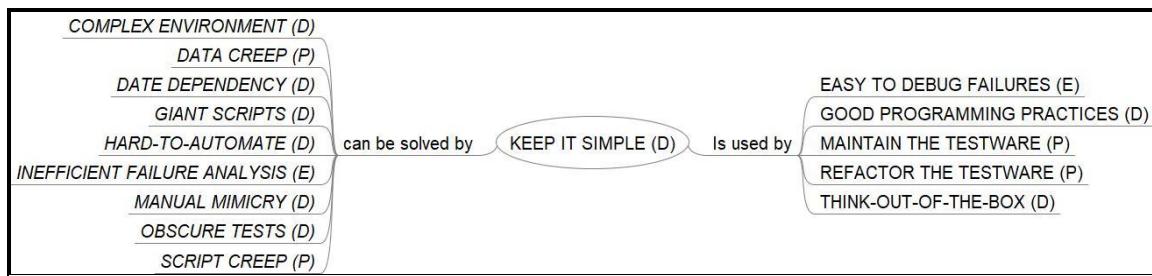


Figure 4.1.35-1 KEEP IT SIMPLE

Pattern summary

Use the simplest solution you can imagine.

Category

Design

Context

This pattern should be adopted for long lasting automation but also when writing disposable scripts.

Description

Use the simplest solution you can imagine. Simple solutions are easier to understand and can thus be reused more often.

Use this pattern especially when automating manual tests: often automation tools let you easily do things (like for instance getting information directly from a database) that a manual tester cannot replicate.

Implementation

One of the rules of extreme programming is simplicity, that is one should implement just what one needs now, no more and no less. The reasoning behind this rule is that mostly when you implement something because you may need it later, the later never comes up or if it does, it is completely different from what you expected so that you worked for nothing.

We have already mentioned scripting is also programming, so one should apply this rule also when automating tests.

Recommendations

If you later think of a better solution, then REFACTOR THE TESTWARE. Another pattern to use in this context is THINK OUT-OF-THE-BOX.

Issues addressed by this pattern

COMPLEX ENVIRONMENT

DATA CREEP

DATE DEPENDENCY

HARD-TO-AUTOMATE

INEFFICIENT FAILURE ANALYSIS

*MANUAL MIMICRY
OBSCURE TESTS
SCRIPT CREEP*

4.1.36. KEYWORD-DRIVEN TESTING (Design Pattern)

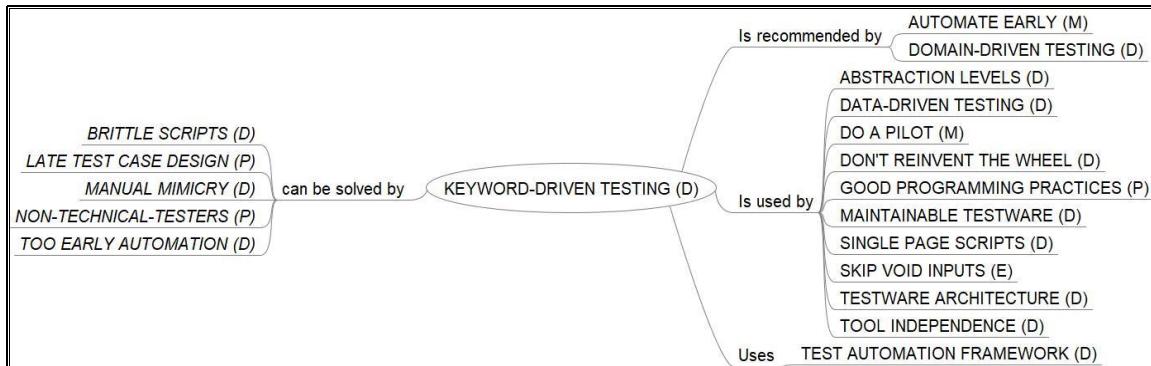


Figure 4.1.36-1 KEYWORD-DRIVEN TESTING

Pattern Summary

Tests are driven by keywords (also called action words) that represent actions of a test and may include input data and expected results.

Category

Design

Context

This pattern is appropriate:

- When you want to write test cases that are practically independent from the Software under Test (SUT). If the SUT changes, the functionality behind the keyword must be adapted, but most of the time the test cases themselves are still valid.
- When testers should be able to write and run automated tests even if they are not adept with the automation tools.
- If you want testers to start writing test cases for automation before the (SUT) is available to test.

The pattern is not appropriate for very small-scale or one-off automation efforts.

Description

Keywords are the verbs of the language that the tester uses to specify tests, typically from a business or domain perspective. A keyword specifies a sequence of actions together with any required input data and/or expected results.

Keyword	Input Data				Expected Results	
	1	2	...	n	1	...
Action 1	Data 1	Data 1.2
Action 2	Data 2	Data 2.2	...	Data 2.n	Result 1	...
...

Keywords are most powerfully used at a high level, representing a business domain. Different domains would have different keywords. High level keywords for an insurance application, for example, might include "Create New Policy", "Process Claim" or "Renew Policy". High level keywords for a mobile phone, for example, might include "Make Call", "Update Contact" or "Send Text Message". There may be some keywords that are common across more than one domain, particularly at lower levels, such as "Log In" and "Print Page".

Implementation

Each keyword is processed by an associated script, which may call other reusable scripts from a library. A keyword script can be written in a common coding language, as calls to other keywords or in the scripting language of the tool. It is a good idea to keep the tool-specific scripts to a minimum as this will help to reduce script maintenance costs. The keyword script reads and processes the input data for the keyword, and/or checks the expected output. The implementation of the architecture of the keywords is often referred to as a TEST AUTOMATION FRAMEWORK, which determines the choice of scripting language and how composite keywords are defined.

Potential problems

Take care that the keywords describe a clear and cohesive action. If the keywords build the “words” for a domain specific language (as in DOMAIN-DRIVEN TESTING), your testers will easily be able to write automation test cases and will be able to start even before the SUT is available for running tests.

Another problem can come up when you need too many parameters (input data) for a given keyword: it becomes unwieldy if you have to scroll repeatedly in order to see or input all the parameters. Break the unwieldy keyword into shorter independent functions (which can call and be called from the others).

Issues addressed by this pattern

BRITTLE SCRIPTS
LATE TEST CASE DESIGN
MANUAL MIMICRY
NON-TECHNICAL-TESTERS
TOO EARLY AUTOMATION

4.1.37. KILL THE ZOMBIES (Process Pattern)



Figure 4.1.37-1 KILL THE ZOMBIES

Pattern summary

Regularly delete or archive test cases, scripts or test data that are not used. Make sure that tests are actually testing something.

Category

Process

Context

This pattern should be applied to maintain long lasting automation.

Description

Regularly archive test cases, scripts or test data that are not used. Make sure that tests are actually testing something.

Implementation

Check regularly for test cases, scripts or data that aren't used any longer and remove them from the system.

Be suspicious if tests seem to be passing too easily. If you are "taking over" existing automated tests from someone else, make sure you know what all of the tests are actually doing.

Write some scripts or use monitoring tools to find out what is going on.

Potential problems

If your tool doesn't support you here, use the "find in files" function of a simple text editor to check what is actually still in use.

Issues addressed by this pattern

DATA CREEP

OBSCURE TESTS

SCRIPT CREEP

4.1.38. KNOW WHEN TO STOP (Management Pattern)

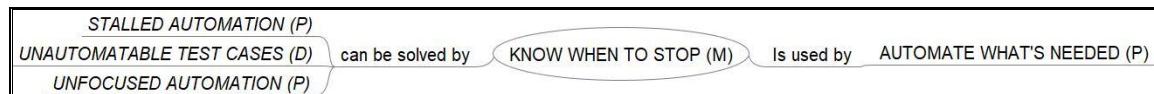


Figure 4.1.38-1 KNOW WHEN TO STOP

Pattern Summary

Not all test cases can or should be automated.

Category

Management

Context

This pattern is most important if you want to implement test automation efficiently, so it is useful even if you just want to write disposable scripts.

Description

Not all test cases can or should be automated. The following test types are better handled manually:

- User interface and usability tests.
- Tests that run only once in a while.
- Tests for a system that is still very unstable.
- Tests that take too long to automate (not worth the effort).

When you are automating tests (just as when you are testing), there will be many things that you could automate (test) - a virtually infinite list. But your time will always be limited, so it is getting the balance right between not automating enough and losing benefits you could have and automating too much and getting decreasing returns from additional automated tests. When to stop is at the point where more tests wouldn't give you enough additional value.

Some tests are very difficult to automate, and the time spent on automating them would be better spent testing this particular thing manually and automating other things. For example, what if you are testing "Captcha" - the wavy letters that you need to type in to a web site to prove that you are a human being? The point of Captcha is that it shouldn't be possible for a computer to do this, so if you could automate the tests for it, it would actually prove that it didn't work! Yes, there are ways to test Captcha automatically, but if this needs to be tested only a few times, it would be better to test it manually, and automate things that are more straightforward.

Implementation

If you automated the most important tests first, in order of value, then whenever you stop, you will have automated the best tests that you could. So, prioritize what tests are most important to automate.

Issues addressed by this pattern

STALLED AUTOMATION

UNAUTOMATABLE TEST CASES

UNFOCUSSED AUTOMATION

4.1.39. LAZY AUTOMATOR (Process Pattern)

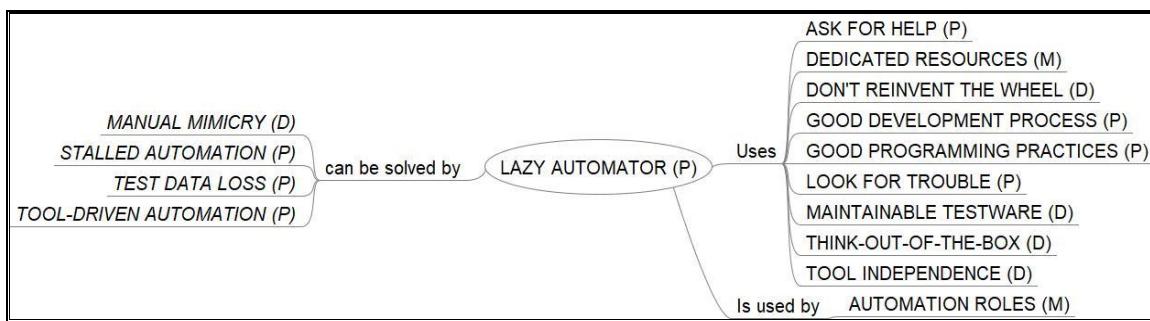


Figure 4.1.39-1 LAZY AUTOMATOR

Pattern summary

Lazy people are the best automation engineers.

Category

Process

Context

This pattern helps you start and keep efficient test automation and should be applied for big and small automation efforts.

Description

Why are lazy people the best automation engineers? They are too "lazy" to do the same boring job over and over again. Instead they devise ways to shortcut it with help from scripts or tools and profit from common good practices. Whenever they find themselves (or someone else) repeating a task, especially if it is "mechanical" in nature, it could probably be automated (provided it doesn't take too much effort). For a "lazy automator" the focus is not only on building efficient testware but also on always keeping in mind that this testware has to be maintainable. Will he or she be able to update this testware six months from now with only a minimum of effort?

Implementation

Since what you have to automate varies from application to application, there can be no general solution; what is important is to be aware that there usually is a more efficient way to do something.

The following patterns are crucial in making your automation process and your automation testware efficient:

- ASK FOR HELP: don't waste time trying to do everything by yourself. If somebody knows better, do not hesitate to ask for help.
- DON'T REINVENT THE WHEEL: if a problem has already been solved, then make the solution yours and go on from there.
- THINK OUT-OF-THE-BOX: The best automation solutions are often the most creative.
- Design MAINTAINABLE TESTWARE, which means that you should adopt GOOD PROGRAMMING PRACTICES and a GOOD DEVELOPMENT PROCESS.

- See that you can draw on DEDICATED RESOURCES: if the test automation team members are jumping from one project to another, they will not be as effective as they could be. A lot of time will be wasted, and not only for the test automation project, just trying to remember where they stopped and what comes next. Also, if you don't have exclusive use of the machines you need, it may be much more difficult to get the initial conditions right, because other developers or testers will corrupt your data over and over again.
- LOOK FOR TROUBLE: Keep your eyes open for possible problems; the sooner you spot them, the easier it will be to solve them.
- Try to adopt TOOL INDEPENDENCE: sooner or later your SUT will change in a way that renders your current test tool obsolete. Or your tool will not be supported any longer and the next tool, even if developed by the same vendor, will not be compatible with the old one. If you keep your dependencies on the tools to a minimum, you will be able to face such events without having to start again from scratch.

Potential problems

We say that the automator should be "lazy", but only in the sense of not wanting to repeat actions that could be automated.

Sometimes in trying to achieve that goal, automators might become so fixated on trying to automate something, that they don't realise how much time they have spent on it, and this can be wasteful. It is only a net gain if it can be automated at a reasonable cost (in effort and time).

Issues addressed by this pattern

MANUAL MIMICRY

STALLED AUTOMATION

TEST DATA LOSS

TOOL-DRIVEN AUTOMATION

4.1.40. LEARN FROM MISTAKES (Process Pattern)

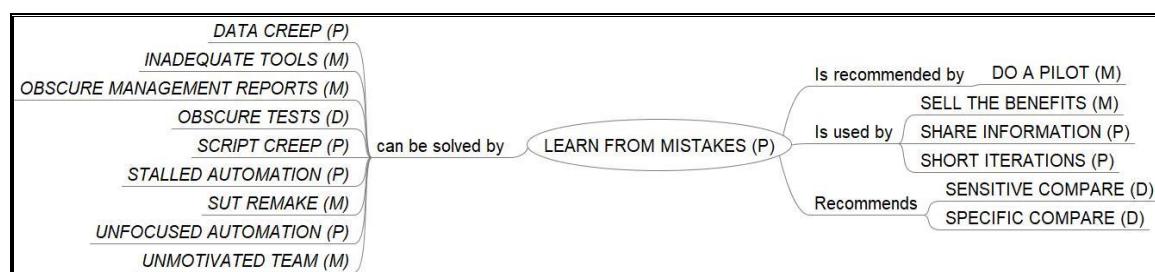


Figure 4.1.40-1 LEARN FROM MISTAKES

Pattern summary

Use mistakes to learn to do better next time.

Category

Process

Context

This pattern is appropriate when a previous test automation project failed (you can also learn from mistakes other people made!) or when the current automation effort is not getting anywhere.

This pattern is also useful when test automation doesn't deliver the results you expect in some area (for instance reporting, tool use, complexity).

Since we don't know anyone who has never made a mistake, we think this pattern is always applicable.

Description

Analyse any failures and deficiencies to find out what went wrong so that next time you don't make the same mistakes.

Implementation

Get all the concerned people together (managers, testers, the test automation team etc.) and examine what was good and what went wrong. Discuss how you could do better next time and adapt accordingly. This may affect processes, procedures, responsibilities, training, standards, etc.

This is not something that you do only once; regularly look back at your recent or past experiences and see how you can improve.

Potential problems

Sometimes what seems to be a disaster can be a useful catalyst for change. When something goes very wrong, people are much easier to convince that something must change, be it the process, the tool, whatever. Be sure to exploit this advantage!

Be very careful to establish a "learning culture" not a "blame culture". If you "point the finger" at individuals, you will only encourage better hiding of mistakes next time. Mistakes are very rarely down to one person; the context and culture also contribute, so everyone is responsible if mistakes are made.

Issues addressed by this pattern

DATA CREEP

INADEQUATE TOOLS

OBSCURE MANAGEMENT REPORTS

OBSCURE TESTS

SCRIPT CREEP

STALLED AUTOMATION

SUT REMAKE

UNFOCUSSED AUTOMATION

UNMOTIVATED TEAM

4.1.41. LOOK AHEAD (Process Pattern)



Figure 4.1.41-1 LOOK AHEAD

Pattern summary

Keep in touch with the tester, automation and development community in order to stay informed about new tools, methods, patterns etc.

Category

Process

Context

This pattern is always useful.

Description

Stay informed about new tools, methods or patterns that can help you make your own automation effort more efficient.

Implementation

There are lots of ways to keep in touch. Here some suggestions:

- Visit testing or development conferences. They usually also feature an expo where the newest tools can be appraised.
- Visit this wiki regularly.
- Visit specialized blogs or forums.
- Look out for new reference books.
- Speak with testers, developers or automators from other companies or departments.

Potential problems

Don't run after every fad!

Issues addressed by this pattern

AUTOMATION DECAY

4.1.42. LOOK FOR TROUBLE (Process Pattern)

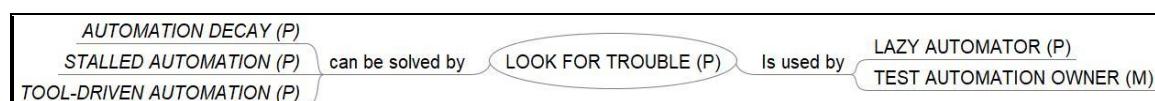


Figure 4.1.42-1 LOOK FOR TROUBLE

Pattern summary

Keep an eye on possible problems in order to solve them before they become unmanageable.

Category

Process

Context

Use this pattern regularly in any context. How often depends on the size of your automation effort or on the agility of your development environment (every day, every build, every iteration, every month and so on).

Description

Control regularly the health of your test automation so that you can solve eventual problems proactively

Implementation

To access how "healthy" your automation is, ask your team regularly and review the measurements you have put in place to see if you are achieving your automation goals. The value of the information lies not in the actual answers or numbers, but in seeing trends, and whether or not you are making progress towards your goals. You should use the information as a gauge that tells you when it would be sensible to go to the Diagnostic.

Here a short list of possible topics to keep an eye on:

- Does everybody on the automation team know at least the basics of automation and of tool use?
- Is management up-to-date on the state of the automation effort?
- Are the automation goals (still) valid?
- Does management (still) support automation?
- Does the current tool (still) drive the SUT (System under Test) to complete satisfaction?
- Are the updates to the automation testware due to changes to the SUT (still) manageable?
- Is the number of automated test cases increasing at the expected rate?
- Are the automation stakeholders satisfied?

Potential problems

Remember: finding out where the problems are just increases frustration if you don't go on and solve them!

Issues addressed by this pattern

AUTOMATION DECAY

STALLED AUTOMATION

TOOL-DRIVEN AUTOMATION

4.1.43. MAINTAIN THE TESTWARE (Process Pattern)

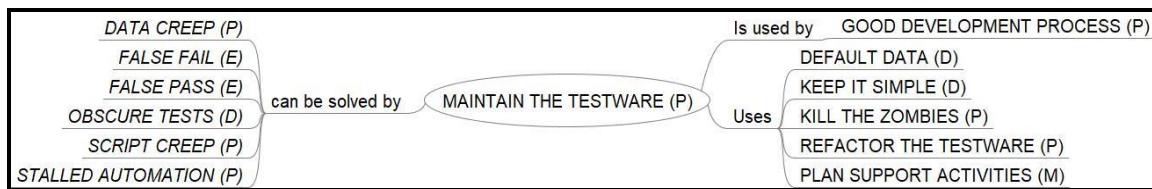


Figure 4.1.43-1 MAINTAIN THE TESTWARE

Pattern summary

Maintain test automation scripts and test data regularly and from the very beginning.

Category

Process

Context

This pattern must be applied if you want your test automation effort to be long lasting. You may not need it for disposable or one-off scripts.

Description

Test automation scripts and test data need to be kept in step with the systems and software that are being tested; as the production software changes, it may cause scripts to fail or use the wrong data. If the testware is not regularly maintained from the very beginning, the automation will fall into disuse and eventually die. The biggest factor contributing to excessive maintenance is the structure of the testware, but constant maintenance is still needed throughout the useful life of the automation.

Implementation

Here is some advice for keeping on top of testware maintenance:

- If you find an error in the test automation scripts, correct it as soon as possible.
- If tests are too complicated, simplify them. (KEEP IT SIMPLE).
- If a test case doesn't pass any longer due to changes in the Software Under Test (SUT), check if it is still valuable and if it is, adjust it accordingly (scripts or data). If it isn't, KILL THE ZOMBIES.
- Verify occasionally that the returned results really are correct. After some time FALSE PASS has a way of slouching in.
- REFACTOR THE TESTWARE, possibly at a time when code is not changing (you can't refactor tests and change code at the same time!)
- Run all your tests regularly. You will quickly notice what is obsolete and what is not.
- PLAN SUPPORT ACTIVITIES: Plan for regular maintenance as part of the ongoing test automation effort.
- Use DEFAULT DATA when your tests need a lot of data, but only a tiny part of it is really pertinent to the test case.

Potential problems

You may find that certain maintenance tasks are taking an increasing amount of time. In this case it may be time to set aside time for a major re-structuring and **REFACTOR THE TESTWARE**. Refactoring is like a spurt of dedicated and concentrated maintenance but carried out so that future maintenance becomes easier after the restructuring.

It may be hard to justify the time taken to maintain the testware when deadlines are tight, and resources are in short supply. But without maintenance, your automation will die, just as a plant dies without water.

Issues addressed by this pattern

DATA CREEP

FALSE FAIL

FALSE PASS

OBSCURE TESTS

SCRIPT CREEP

STALLED AUTOMATION

4.1.44. MAINTAINABLE TESTWARE (Design Pattern)

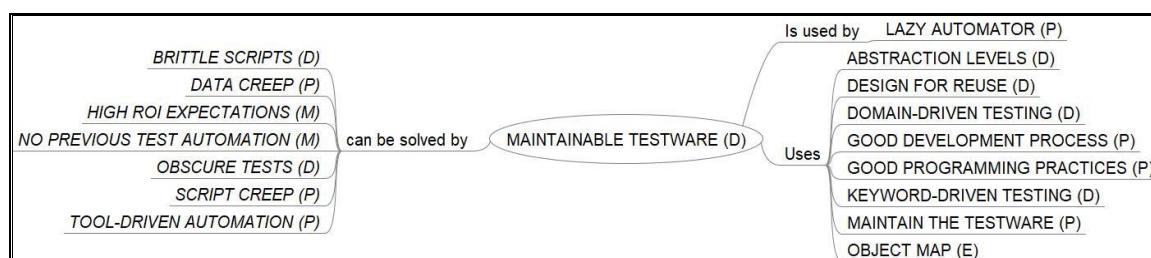


Figure 4.1.44-1 MAINTAINABLE TESTWARE

Pattern summary

Design your testware so that it does not have to be updated for every little change in the Software Under Test (SUT).

Category

Design

Context

This pattern is applicable when your automated tests will be around for a long time, and/or when there are frequent changes to the SUT.

This pattern is not applicable for one-off or disposable scripts.

Description

Identify the costliest and/or most frequent maintenance changes and design your automation to cope with those changes with the least effort. When adjustments really are necessary, then they should be relatively easy to implement. For example, if objects are frequently renamed, construct a translation table from the

name you want to use in the tests, and put in whatever the name of the object is for the current release of the SUT. (OBJECT MAP).

Implementation

Some suggestions:

- There are many options to make and keep the testware maintainable, but to adopt a GOOD DEVELOPMENT PROCESS and GOOD PROGRAMMING PRACTICES is a very good bet: what works for software developers works for test automation just as well!
 - For example, if your scripts are mainly "stand-alone" without much re-use, and automators are frequently re-inventing similar or even duplicated scripts or automated functions, then GOOD PROGRAMMING PRACTICES are needed, particularly DESIGN FOR REUSE and OBJECT MAP.
- Implement DOMAIN-DRIVEN TESTING: test automation works best as cooperation between testers and automation engineers. The testers know the SUT but are not necessarily adept in the test automation tools. The automation engineers know their tools and scripts but may not know how best to test the SUT. If the testers can develop a domain-specific language for themselves to use to write the automated test cases, the automation engineers can implement the tool support for it. In this way they will each be doing exactly what they do best. The advantage for the automation engineers is that in this way the testers can do some of the maintenance of the tests themselves leaving the engineers more time to refine the automation regime, and to solve technical maintenance problems.
 - For example, if you have structured and reusable scripts, but there is a shortage of test automators to produce the automated tests (or they are short of time), this pattern gives the test-writing back to the testers, once the automators have constructed the infrastructure for the domain-based test construction. Other useful patterns are ABSTRACTION LEVELS and KEYWORD-DRIVEN TESTING.
- Don't forget to actually MAINTAIN THE TESTWARE: maintainability alone is not enough, it must actually *be* maintained.

Potential problems

Don't wait too long to build maintainable testware - this is best thought of right at the beginning of an automation effort (although it is also never too late to begin with improvements).

Issues addressed by this pattern

BRITTLE SCRIPTS

DATA CREEP

HIGH ROI EXPECTATIONS

NO PREVIOUS TEST AUTOMATION

OBSCURE TESTS

SCRIPT CREEP

TOOL-DRIVEN AUTOMATION

4.1.45. MANAGEMENT SUPPORT (Management Pattern)

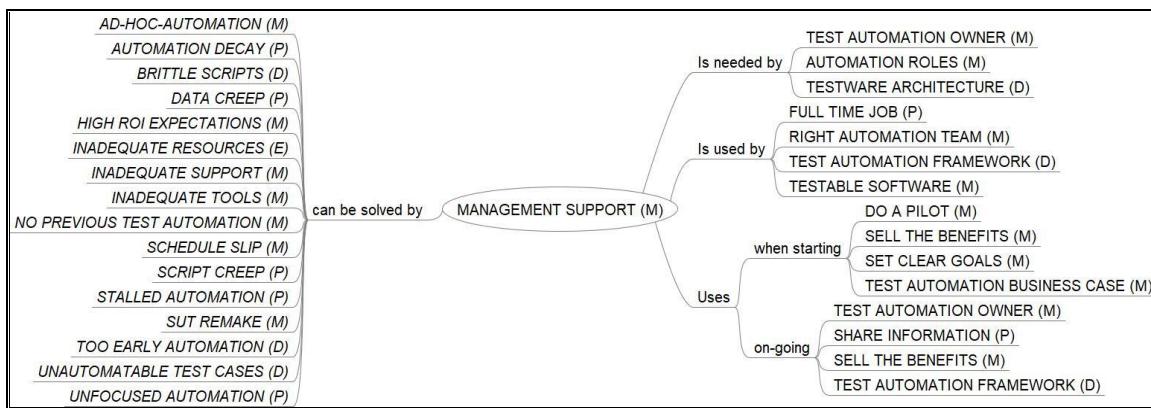


Figure 4.1.45-1 MANAGEMENT SUPPORT

Pattern summary

Earn management support. Managers should only support sound and well-reasoned activities, so we need to work at selling the idea initially and then keep them up-to-date with progress and issues.

Category

Management

Context

This pattern is applicable when test automation is intended to be used by many people within an organisation.

This pattern is not applicable for one person beginning to experiment with a tool to see what it can do.

Description

Many issues can only be solved with good management support.

When you are starting (or re-starting) test automation, you need to show managers that the investment in automation (not just in the tools) has a good potential to give real and lasting benefits to the organisation.

In an ongoing project, inform regularly on the status and draw special attention to any success or return on investment. You still need to have good communication and a good level of understanding of current issues from management. Sometimes a single incident can be more convincing than a large set of numbers, for example if a recurring bug is found by an automated regression test for a user that had complained about this same bug twice before.

Implementation

Some suggestions when starting (or re-starting) test automation:

- Make sure to SET CLEAR GOALS. Either review existing goals for automation or meet with managers to ensure that their expectations are realistic and adequately resourced and funded.
- Build a convincing TEST AUTOMATION BUSINESS CASE. Test automation can be quite expensive and requires, especially at the beginning, a lot of effort.
- A good way to convince management is to DO A PILOT. In this way they can actually "touch" the advantages of test automation and it will be much easier to win them over.
- Another advantage is that it is much easier to SELL THE BENEFITS of a limited pilot than of a full test automation project. After your pilot has been successful, you will have a much better starting position to obtain support for what you actually intend to implement.

Some suggestions for on-going test automation:

- If you don't have one, appoint a TEST AUTOMATION OWNER to monitor test automation health.
- SHARE INFORMATION about the automation. People soon forget about the benefits that have been achieved, so it is good to keep reminding them.
- If you have *INADEQUATE SUPPORT*, you may have to free some people from their current assignments.
- If you have *INADEQUATE TOOLS*, you may need to invest in new tools or build or revise your TEST AUTOMATION FRAMEWORK.
- In these cases, you may need to SELL THE BENEFITS in order to convince management that the investment will be worthwhile.

Potential Problems

It is almost equally important to set realistic expectations about what the test automation project can deliver. *UNREALISTIC EXPECTATIONS* can lead to disappointment and frustration and you can lose management support just when you need it most.

Another problem that can arise is that the manager talks about supporting you and claims to support your efforts. But when you need to take some additional time or use additional resources, then "sorry, they are not available". This is not true support, but "lip service".

It is also possible to inadvertently set *UNREALISTIC EXPECTATIONS* by being overly enthusiastic about what can be accomplished early on in automation. It can be easy to show good results when you haven't yet encountered any of the problems that will occur later, such as the cost of maintaining the automated tests when the software under test is changed.

Issues addressed by this pattern

AD-HOC AUTOMATION

AUTOMATION DECAY

BRITTLE SCRIPTS

DATA CREEP

HIGH ROI EXPECTATIONS
INADEQUATE RESOURCES
INADEQUATE SUPPORT
INADEQUATE TOOLS
NO PREVIOUS TEST AUTOMATION
SCHEDULE SLIP
SCRIPT CREEP
STALLED AUTOMATION
SUT REMAKE
TOO EARLY AUTOMATION
UNAUTOMATABLE TEST CASES
UNFOCUSSED AUTOMATION

4.1.46. MIX APPROACHES (Management Pattern)



Figure 4.1.46-1 MIX APPROACHES

Pattern summary

Don't be afraid to use different tools or approaches.

Category

Management

Context

This pattern is appropriate when you want to automate as efficiently as possible.

Description

Use a variety of different tools or approaches, as they can complement each other and give you much better results than if you stick to just one tool or approach.

Implementation

- For every task, use the tool that fits best.
- Freeware tools can be just as effective as expensive commercial ones.
- PREFER FAMILIAR SOLUTIONS: if possible chose tools that are already in use in your organization and that are well known.
- AUTOMATE WHAT'S NEEDED: decide individually for each application if it's better to use test automation, manual tests or a mixture of both.

Issues addressed with this pattern

INADEQUATE TOOLS
MANUAL INTERVENTIONS

4.1.47. MODEL-BASED TESTING (Design Pattern)



Figure 4.1.47-1 MODEL-BASED TESTING

Pattern Summary

Derive test cases from a model of the Software Under Test (SUT), typically using an automated test case generator.

Category

Design

Context

The main advantages of model-based testing (MBT) are the systematic test coverage of the model and a significant reduction of the test maintenance effort. In case of changes to the SUT, only a few parts of the model have to be updated, while the test cases are updated automatically by the generator.

Model-based testing can be used for generating GUI test scripts, but also for producing scripts for embedded systems or even manual test instructions. The approach is especially useful for projects with many test repetitions but will not pay off if there are only a few. Also, its efficiency depends on the re-usability of the model components. For instance, if the "Login" sequence is similar for most test cases, it can be defined as a re-usable building block. But if the test cases hardly contain any duplicate sequences, the re-usability will be low, thus increasing the effort for model creation and maintenance.

Description

After creating a test model for the SUT, test data and/or test sequences can be systematically derived from it.

Implementation

Modelling should start as early as possible in the project. Based on the requirements, a rough model can be created and then be refined step by step. Technical details (e.g., information about the affected GUI objects) are only added in the last stage. Particular attention has to be paid to the clearness of the test model and the re-usability of its components. The usage of parametrizable building blocks (sub-sequences) is highly encouraged.

The generation algorithm will search for valid paths through the model until the required coverage criteria are met. Apart from test sequences, some tools can also generate test data using systematic methods like equivalence partitioning or cause/effect analysis.

Potential problems

In many projects, the requirements / design specifications do not have the level of detail required for creating a model. Therefore, often a lot of effort has to be spent to gather the missing information. On the other hand, many weaknesses and

inconsistencies in these documents are detected at an early stage just by creating the model.

Another issue of many test generators is that they only produce sequences of abstract keywords rather than executable scripts, e.g., for test execution tools. In order to avoid additional implementation and maintenance effort, it is important to choose a MBT tool that is able to produce complete scripts for the planned execution tool.

Issues addressed by this pattern

BRITTLE SCRIPTS

OBSCURE TESTS

SCRIPT CREEP

4.1.48. OBJECT MAP (Execution Pattern)



Figure 4.1.48-1 OBJECT MAP

Pattern summary

Declare all the GUI-Objects in the Object Map of the test automation tool. (Or define your own if appropriate).

Category

Execution

Context

This pattern is appropriate if you want your test automation to be long lasting, because your scripts will be easier to read and less dependent on the current tool. This pattern isn't needed if you only write disposable scripts or if your application is exceptionally stable, with object names never or hardly ever being changed.

Description

Declare all the GUI-Objects in the Object Map of the tool (note: this may also be called a GUI map or Object Repository). You can then write your scripts using standardized names. The scripts will be more readable and if you have to migrate to another tool you will not need to change the scripts to allow for new names even if tools seem never to use compatible naming conventions.

Implementation

This is one way to implement TOOL INDEPENDENCE, as your tests will be one layer removed from the specific names in the software.

Some suggestions:

- Ask the developers to use unique names to define the GUI-Objects. If the same objects are used over a number of different screens (object-oriented inheritance) it may be useful to add the screen name to the object name for better recognition.
- When driving the GUI avoid using pixels, screen coordinates or bitmaps to identify graphical objects. Screen coordinates can vary from machine to machine, from operating system to operating system or from browser to browser. Also changes in the Software under Test (SUT) can mean different positions for the GUI-Objects, so your scripts are more likely to break.
- If the SUT runs in different countries, you must also avoid using text labels if you don't want to write new scripts or have to have a separate Object Map for every new language!
- If you can't use the tool's facilities, you can make your own "translation table", in which you have the names you will use for the tests in one column, and the current name used by the software in another column. Then if the software names change, you only have to update your translation table and all your tests will continue to work.

Some suggestions for object naming:

- Internal developer name: if the developers follow a style guide it can be useful to use the same names. In this way you also have the advantage of being able to communicate with the developers in their own "language".
- Label on screen: using the labels on the screen makes it easier for testers to follow what happens in the script.
- Functional term: using the correct terms is a useful way to document what the script is supposed to achieve.

Potential problems

This approach might be perceived as an unnecessary additional complication, but it is needed to gain a level of abstraction from the tool.

Issues addressed by this pattern

NON-TECHNICAL-TESTERS

OBSCURE TESTS

TOOL DEPENDENCY

4.1.49. ONE CLEAR PURPOSE (Design Pattern)



Figure 4.1.49-1 ONE CLEAR PURPOSE

Pattern summary

Each test has only one clear purpose.

Category

Design

Context

Use this pattern to build efficient, modular and maintainable testware
It's not necessary for disposable scripts.

Description

Examples and the resulting tests should have a single clear purpose derived from one business rule.

Implementation

Make sure that each automated test has just one clearly defined job to do. If you have SET STANDARDS, there should be guidelines for this, and how to describe the purpose of the test.

Issues addressed by this pattern

GIANT SCRIPTS

MANUAL MIMICRY

OBSCURE TESTS

4.1.50. ONE-CLICK RETEST (Execution Pattern)



Figure 4.1.50-1 ONE-CLICK RETEST

Pattern summary

Retesting a specific test case should be as easy as one mouse click.

Category

Execution

Context

This pattern helps to efficiently retest previously failed tests.

This works if the system is state-less, i.e. you don't get different results depending on the state of something.

Description

Make the retest of a specific test case as easy as possible. If you have a TEST AUTOMATION FRAMEWORK to support you, you should be able to select the test case and run it literally with one mouse click.

Implementation

If you can PRIORITIZE TESTS, it will be relatively easy to change the priorities to select only the desired test case. If you add another level of prioritization, you can reduce this effort even more (see Example 1).

Recommendations

An extra level of prioritization allows you to use this pattern even if you have *INTERDEPENDENT TEST CASES* (see Example 2).

Issues addressed by this pattern

INEFFICIENT FAILURE ANALYSIS
MANUAL INTERVENTIONS

4.1.51. PAIR UP (Process Pattern)

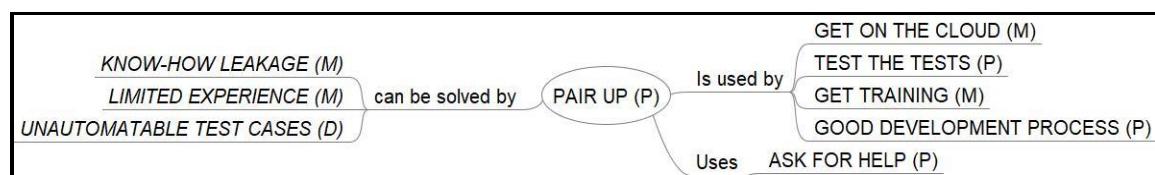


Figure 4.1.51-1 PAIR UP

Pattern summary

Let less experienced team members pair up with more experienced team members.

Category

Process

Context

This pattern can always be applied but is especially useful when team members are expected to learn new skills as fast as possible or to spread knowledge within the team.

Description

Let new or not so knowledgeable team members pair up with more experienced team members when doing some test automation task.

Implementation

In software development it's called pair programming. Two developers sit together at one computer. One writes the code and the other looks on and reviews what the other is doing. They discuss eventual problems together. The one that is less experienced can learn very quickly how the more experienced developer writes code or solves some kind of issue.

In test automation it works the same way: for instance, if a new person on the team doesn't know the tools yet, by pairing he can learn by seeing what the other is doing. He (or she) can ASK FOR HELP directly.

By pairing you can spread know-how in the entire team so that if someone leaves the company, you will not feel the loss as badly.

Pairing testers and automators helps them understand their respective issues.

Pairing is good not only for training, but also to work faster since it implies an immediate review of what is being done: quality is better from the start. It should be used every time non-trivial issues have to be solved.

Potential problems

Managers may see two people working on the same thing as wasteful. You may need to point out the long-term advantages.

It is important that the people working together have a good combination of support for each other and a "critical eye". If they don't get on well, they won't work effectively, but if they are too "nice" they will accept each other's work without challenging it.

Issues addressed by this pattern

KNOW-HOW LEAKAGE

LIMITED EXPERIENCE

UNAUTOMATABLE TEST CASES

4.1.52. PARALLELIZE TESTS (Execution Pattern)



Figure 4.1.52-1 PARALLELIZE TESTS

Pattern summary

In order to save time, run tests in parallel.

Category

Execution

Context

Use this pattern when you have a great number of test to execute in a short time.

Description

Instead of running all your tests sequentially, you break them up in different batches that can run concurrently on different machines.

Implementation

The first step is to write the tests as INDEPENDENT TEST CASES. Another good way to do this is to GET ON THE CLOUD

Potential problems

Be careful not to run the same tests in parallel!

Issues addressed by this pattern

INEFFICIENT EXECUTION

4.1.53. PLAN SUPPORT ACTIVITIES (Management Pattern)



Figure 4.1.53-1 PLAN SUPPORT ACTIVITIES

Pattern summary

Plan time and resources not only for the project goals, but also for support activities.

Category

Management

Context

This pattern is appropriate if you are to have long lasting automation, since you will need specialists both for implementing new automation and for maintaining it.

This pattern isn't appropriate for one-off or disposable scripts, or for short-term one-off automation efforts where support will not be required.

Description

Planning often focuses on initial and major changes, such as acquiring a new tool. However, planning should also be done for the ongoing activities that will be needed to provide continuous support for the automation, after the initial major changes. If such support is not planned, technical debt can build up and can cause serious damage to the test automation.

Implementation

Your planning should already have taken into account the time and effort needed to get started, and you may already have done a pilot. Once the pilot has shown that automation can work for your organisation, there will be ongoing investment needed to continually support the automation.

For example, more people may need to become familiar with the tool; for this you may need to GET TRAINING.

You may need knowledge and advice from specialists for particular technical problems with the tests, so don't be afraid to ASK FOR HELP.

You may also need to work closely with the developers of the Software under Test (SUT) to make sure that the software is testable using the automation, so SHARE INFORMATION about what you need, and also ask what you can do to help them. For example, if you can quickly run some smoke tests, that may benefit the developers, and they will be more likely to cooperate with you.

Although you may not know in advance what support activities will be needed, it is important to anticipate that some will be needed!

In your business case don't forget to plan resources and time for:

- Testing and bug fixing of the automated tests.
- Refactoring (SUT and the testware).
- Training, both in the tool itself and in the regime that you put in place for effective automation (e.g. local naming conventions).
- Support from developers, database specialists, network administrators and others.

Your managers may not appreciate what is needed or what you are spending time on. At every presentation show your managers why you need this extra time and resources. Calculate the costs and the technical debt of ignoring this issue.

Potential problems

- You need more time than was planned to automate test cases, because you are not well acquainted with the tool.
- You need support from testers because the manual test cases are not documented well enough or are not 'automatable'.
- You need support from IT specialists (e.g. database managers), but they never have time.
- You need support from the developers of the SUT, but they are always busy
- The SUT is not really "testable".
- Your test suites find lots of bugs, but there is no time to fix them or the fixing takes longer than expected.
- You would like to refactor your testware, but you never have the time or the resources to do it.

Issues addressed by this pattern

HIGH ROI EXPECTATIONS

INADEQUATE SUPPORT

NO PREVIOUS TEST AUTOMATION

SCHEDULE SLIP

UNAUTOMATABLE TEST CASES

UNMOTIVATED TEAM

4.1.54. PREFER FAMILIAR SOLUTIONS (Management Pattern)

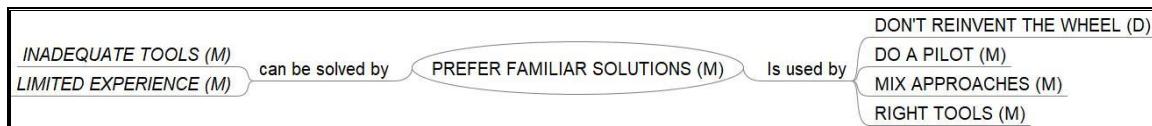


Figure 4.1.54-1 PREFER FAMILIAR SOLUTIONS

Pattern Summary

Given the same benefits prefer a tool or a scripting language that is already known in-house.

Category

Management

Context

This pattern should be applied when you are selecting a new tool or script language

Description

Given the same benefits prefer a tool or a script language that is already known in-house. In this way you'll have the advantage that your testers or automation engineers are already familiar with the tool or scripting language and don't need as much time and effort to become fully productive

Implementation

Look up what is already in use in your company. If you get the features you need, then definitely keep using the tool or scripting language

Recommendations

If you do select a new tool or scripting language and have to find new people, then you should choose what is most popular, because it will facilitate the search for skilled personnel

Issues addressed by this pattern

INADEQUATE TOOLS

LIMITED EXPERIENCE

4.1.55. PRIORITIZE TESTS (Execution Pattern)



Figure 4.1.55-1 PRIORITIZE TESTS

Pattern summary

Assign each test some kind of priority in order to be able to select easily the ones that should be run.

Category

Execution

Context

Use this pattern when you may not be able to run all of the tests that you have in your automated portfolio.

Use this pattern to be selective about tests to run when something has changed.

Description

Assign each test some kind of priority, based on the criteria that are important at the time. The priority may be related to recent changes, the most critical user-facing aspects of the SUT (System Under Test), a recent change to some related system, or any other factor. The priority tests to run now may be different to the priority of the tests that you will want to run tomorrow.

You should be able to easily select subsets of tests to run, based on the priority assignment.

Implementation

- Group all the test cases that test the same functionalities of the SUT in the same test suite. In this way you can test different parts at different times or concurrently.
- Reserve a parameter on every test case for the priority and configure your tool or TEST AUTOMATION FRAMEWORK so that you can select the priority of the test cases to be run. Common values are:
 - Critical
 - Important
 - Average
 - Minor
- Other categories can also be used to select which tests to run using a TEST SELECTOR, such as
 - all the tests that failed last time
 - the tests for a particular function
 - Joe's tests
 - high level smoke tests

Issues addressed by this pattern

INEFFICIENT EXECUTION

INFLEXIBLE AUTOMATION

4.1.56. READABLE REPORTS (Design Pattern)

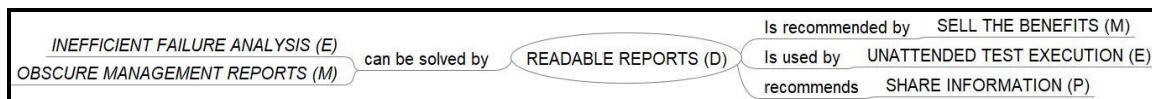


Figure 4.1.56-1 READABLE REPORTS

Pattern summary

Design the reports to be easy for the recipient to read and understand.

Category

Design

Context

Use this pattern for efficient and long-lasting automation.

You will not need it for a one-time solution.

Description

Reports should at a glance show if a test passed, but they must also be helpful in finding why a test failed. Reports must also deliver metrics for management, for instance trends.

What gets reported should be related to the goals and objectives of the automation; hopefully you have already SET CLEAR GOALS.

Implementation

Design the reports in different levels depending on who will read them. Testers or developers will need detailed information to be able to resolve failures but will not want to waste time with passed tests. Managers will need more statistical kinds of data, depending on their goals and what they want to monitor.

At first, you may be reporting mainly on the progress of starting automation; later on, when the automation is more established, you should report on the health of the automation (such as reducing costs and increasing benefits).

Recommendations

SHARE INFORMATION to find out what each recipient needs and wants.

Issues addressed by this pattern

INEFFICIENT FAILURE ANALYSIS

OBSCURE MANAGEMENT REPORTS

4.1.57. REFACTOR THE TESTWARE (Process Pattern)

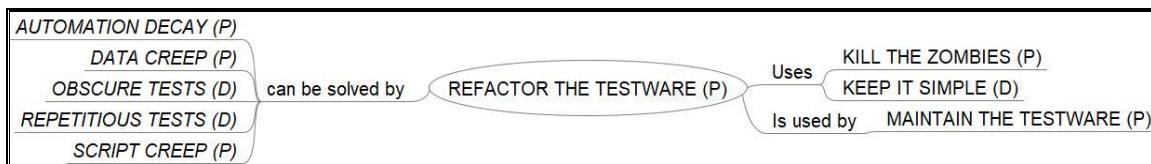


Figure 4.1.57-1 REFACTOR THE TESTWARE

Pattern summary

Testware must be refactored regularly just as application code.

James Tony: Tests have technical debt too! It may be harder to measure than the technical debt of the code being tested but cutting corners when writing tests can lead to a build-up of technical debt that can slow you down to a crawl in the future.

Category

Process

Context

Use this pattern for long lasting automation projects in order to keep maintenance costs low. For short lived solutions you will not need it.

Description

Refactoring means that scripts and test data should be checked regularly:

- Are the automated test cases still valid and useful?
- Can the automation scripts be improved?
- Can similar scripts be merged?
- Is the data up to date?
- Is the documentation up to date?
- Is the technical implementation as efficient as it can be? For example, wait times, communication methods etc.

Implementation

After you have determined what should be refactored schedule when to execute the updates and control that they get done:

- KILL THE ZOMBIES
- Improve the scripts (KEEP IT SIMPLE).
- Upgrade test data to the current release.
- Remove or merge duplicate scripts so that you only have to maintain one version.
- Check that you are using the RIGHT INTERACTION LEVEL.

Potential problems

Refactoring will often be postponed when a project is running out of time. Be careful not to build up too much technical debt

Issues addressed by this pattern

AUTOMATION DECAY

DATA CREEP

OBSCURE TESTS

REPETITIOUS TESTS

SCRIPT CREEP

4.1.58. RIGHT INTERACTION LEVEL (Design Pattern)

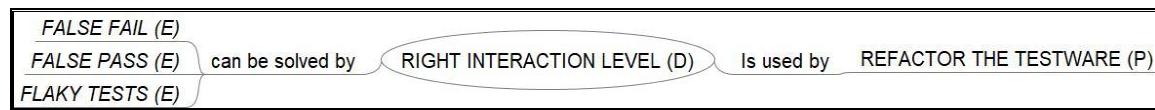


Figure 4.1.58-1 RIGHT INTERACTION LEVEL

Pattern summary

Be aware of the interaction level of the test approach on the Software Under Test (SUT) and its risks (intrusion).

Category

Design

Context

This pattern is useful when test automation on the SUT is started, or changed. The level of interaction can still be influenced.

Description

It is important to realize that there are different ways/levels to interact with the SUT. The different levels reflect different implementations, but also different risks on false positives and false negatives. High intrusiveness of the SUT increases the risk of false test results.

Implementation

When automating test cases, it must be chosen in which way to interact with the SUT. This can be done on multiple levels, and this has impact on the intrusiveness of the test cases. Some examples:

- Automating test cases in the same way as the end user would use the system (via the same interfaces). The SUT is not adapted in this approach. This can be done by testing via hardware interfaces of the SUT (e.g. via USB or TCP/IP interfaces, see also chapter 13 of Experiences of Test Automation). Especially in embedded environments (where software interacts with other disciplines) this is a typical approach. Although this approach can be quite complex and expensive, the intrusion of the test cases on the SUT is (almost) zero, resulting in reliable test results (assuming the test cases have been implemented correctly): the SUT will

not behave differently because of the test approach. This level is rarely used for software only applications.

- User Interface (UI) automation: when performing test automation via the GUI, the environment of the SUT is changed in order to allow this way of testing: a specific tool, or libraries need to run on the SUT (this will affect the timing of the SUT). Although the user is simulated realistically, the level of intrusion is higher than in the previous point, increasing the risk of false positives and false negatives.
- Test automation is often performed on the API (Application Programming Interface), this way it is quite easy to implement automated test cases. Often dedicated test-interfaces are implemented to support the test automation approach (Design for Testability). Although this way of testing is very powerful and efficient, the level of intrusion is very high: the SUT is changed for testing purposes. Failures may be found, that are unreproducible by end-users. These failures are caused by test approach. Test cases do not necessarily reflect realistic user behavior.

Which level of interaction to use should be a conscious choice when test automation starts (any level can be a good choice) considering also the risks of this choice, not only the benefits.

Potential Problems

Different levels of interaction result in different levels of intrusion. See above for examples.

Issues addressed by this pattern

FALSE FAIL

FALSE PASS

FLAKY TESTS

4.1.59. RIGHT TOOLS (Management Pattern)

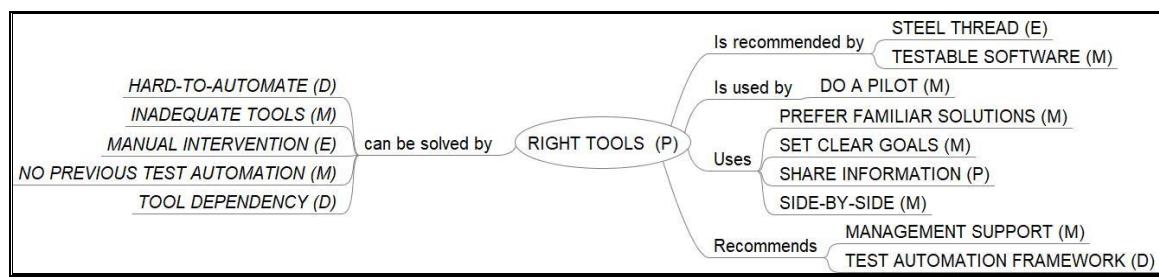


Figure 4.1.59-1 RIGHT TOOLS

Pattern summary

Select the right tools for your application, environment, people, budget and level of testing.

Category

Management

Context

This pattern is appropriate when your automated tests will be around for a long time, and/or when you are beginning test automation and want to "get off on the right foot".

This pattern is not appropriate for one-off or disposable scripts, or when you are experimenting with different tools just to see what they do.

Be aware that having "the right tool" is not all that important, as long as a tool is workable in your context. The other factors involved in test automation are usually much more important and influential than the choice of tool.

Description

In order to select the right tools you must first of all SET CLEAR GOALS as to what you intend to achieve with them and what not. To do this, you must first SHARE INFORMATION with testers and developers to find out what they actually need from automation. Testers and especially managers can have false expectations: things that would be easy to automate are considered so difficult that they are not even mentioned and things that are really difficult to automate are believed to be just a couple of clicks away!

If you are just starting with test automation then your goals will be just to find some tools that can work with your Software Under Test (SUT).

If you are changing to a different tool, an additional important criterion will be how easily you can migrate your tests from your old tool to the new one.

Implementation

Depending on your budget you can select open source, freeware or commercial tools. There are some crucial issues to consider:

- The tool must be able to do what you acquired it for. For instance a tool to run automation from the GUI must recognize all the objects on your screens to drive your application. The following **Score Card*** can help you check that the tool features all the functionality that you need
 - Platform Support – Support for multiple operating systems, tablets & mobile.
 - Technology Support - "multi-compiler" vs. "compiler-specific" test tools.
 - Browser Support - Internet Explorer, Firefox, Google Chrome or any other browser based on web browser controls.
 - Data Source Support - obtain data from text and XML files, Excel worksheets and databases like SQL Server, Oracle and MySQL.
 - Multi-Language Support - localized solutions supporting Unicode.
 - Test Type Support - functional, non-functional and unit (i.e. nUnit & MSTest).
 - Test Approach Support – i.e. Hybrid-Keyword/Data-Driven testing.

- Results & Reporting Integration – including images, files, databases, XML documents.
- Test Asset / Object Management - map an object not only by its caption or identifier.
- Class Identification – GAP analysis of object classes (generic / custom) and associated methods capabilities based on complexity, usage, risk, feasibility and re-usability.
- Test Scenario Maintenance – manual effort (XPATH/regular expressions), self-maintaining (descriptive programming/fuzzy logic) or script less (DSLs).
- Continuous Build & Integration / Delivery Integration – with build & delivery solution.
- Future proofing – external encapsulation of test assets & associated meta data (XAML/xPDL), expandability (API/DLL/.NET), HTTP/WCF/COM/WSD and OCR/IR.
- License, Support & Maintenance Costs – pricing policy along with any hidden costs.
- Most of the commercial tools on the market have demo versions which you can try on your application. Commercial tool vendors offer workshops to prove that their tool is “just right” for you.
- If you get an open source tool, the tool will most likely be maintained and supported by the open source community, although some tools do have commercial companies offering support.

Some questions to ask:

- How easy is it to learn to use the tool? Is it possible to get training?
- How widespread is the script language of the tool? It will be easier to get skilled people for a more popular script language.
- How easy is the migration of your testware from an old tool to the new one?
- How well is the tool supported? How long does the vendor plan to support it? Does the vendor plan to introduce a new tool? If yes, will it be compatible with the old one?
- How high is the probability that with new releases the tool will not support your application as well or not at all (tool regression)?
- How easy is it to compile a usage list?

To find which tools are available on the market you can:

- Search the internet.
- Ask in test automation forums what tools other testers use. Then ask what other testers have experienced with the tools that you are considering.
- Attend a conference or seminar that includes a expo with tools. You can get a quick demo at the stand and get an initial idea about a tool.
- Engage a consultant.

Once you have some good candidates try them SIDE-BY-SIDE.

Finally don't forget to PREFER FAMILIAR SOLUTIONS if there are already workable tools in the company, you will save costs and effort because you will profit from the already available experience

Potential problems

- Before you start make sure you have MANAGEMENT SUPPORT. You will need it to get the time and resources necessary to select and introduce a new tool.
- Remember that the objective of test automation is to support testers. Any kind of tool that helps testers do their job better is a test automation tool!
- Set priorities regarding what should be automated first. Since the team members need time to learn to use a tool, you don't need all the tools at the same time, you could end up paying for licenses that you'll not be using (yet)!
- How much effort you will need to change tools depends primarily on the test automation architecture that has been or will be implemented. If your current architecture doesn't support well the changing of the tool, then this is the right time to change your automation structure and implement a TEST AUTOMATION FRAMEWORK.

Issues addressed by this pattern

HARD-TO-AUTOMATE

INADEQUATE TOOLS

MANUAL INTERVENTIONS

NO PREVIOUS TEST AUTOMATION

TOOL DEPENDENCY

4.1.60. SELL THE BENEFITS (Management Pattern)

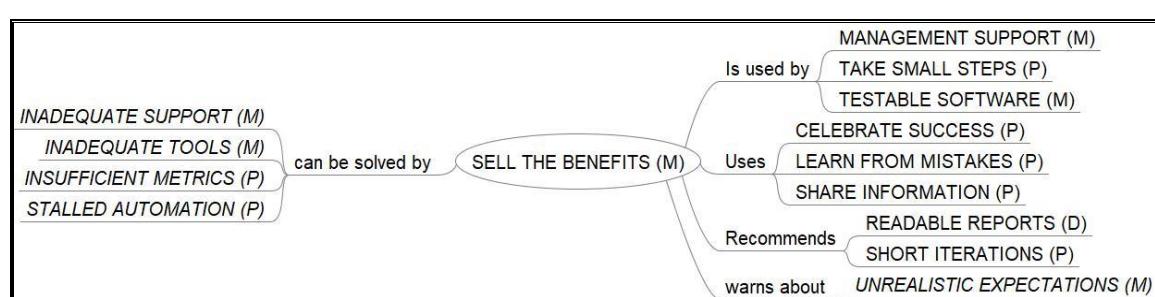


Figure 4.1.60-1 SELL THE BENEFITS

Pattern summary

Inform management before and during test automation about achievable and achieved benefits.

Category

Management

Context

Apply this pattern if you need support from management, testers or developers. You don't need it if you only plan to write some disposable scripts

Description

Regardless what goals you are pursuing, you will not get support if you cannot illustrate the advantages your actions will bring. Inform management and the parties involved before the automation gets started and regularly as long as the automation lives. Let them participate both in your success and in your failures. Success will be a boon for next time (people will know that you deliver what you promised), but even failure can be helpful to avoid similar problems in the future (LEARN FROM MISTAKES).

Implementation

If needed, before the automation starts, prepare a business plan to show management "with numbers" what ROI test automation would bring.

Even if you don't show ROI, outline what the benefits would be from automation. Make sure to distinguish benefits and objectives of automation as separate from testing.

Take care to get a clear picture of the problems you want to solve. Include the involved parties in the search for a solution. When one has participated in a decision he or she will be much more willing to implement it and, just as important, help you explain its benefits to management, testers and developers.

While the project is running SHARE INFORMATION and CELEBRATE SUCCESS.

Use SHORT ITERATIONS to be able to give frequent feed-back to your managers. Be sure to design READABLE REPORTS.

Potential problems

Be careful that you aren't too successful! It is easy to describe the benefits a bit too enthusiastically, and managers may tip into *UNREALISTIC EXPECTATIONS*

Issues addressed by this pattern

INADEQUATE SUPPORT

INADEQUATE TOOLS

INSUFFICIENT METRICS

STALLED AUTOMATION

4.1.61. SENSITIVE COMPARE (Design Pattern)

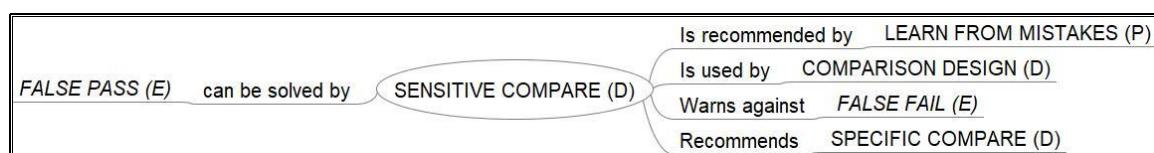


Figure 4.1.61-1 SENSITIVE COMPARE

Pattern summary

Expected results are sensitive to changes beyond the specific test case.

Category

Design

Context

This pattern is applicable when your automated tests will be around for a long time, and/or when there are frequent changes to the SUT.

This pattern is not applicable for one-off or disposable scripts.

Description

The expected results compares a large amount of information, more than just what the test case might have changed. For example, the comparison of an entire screen or window (possibly masking out some data). Sensitive tests are likely to find unexpected differences and regression defects.

Implementation

Implementation depends strongly on what you are testing. Some ideas:

- Extract from a database the entire tables touched by processing the test case.
- Check the whole log and not only the parts directly pertaining to the test case.
- On the GUI check all the objects on each page.

If you are checking the whole of a window or screen, you may want to mask out data that you are not interested in, such as the date and time of the test. Otherwise, the date/time would be a difference shown up by the comparison, but you don't want that information!

Potential problems

If all your test cases use this pattern you would probably often get *FALSE FAIL!*.

It makes sense to have at least some test cases using this pattern, for example in a smoke test or high-level regression test. Other tests should use SPECIFIC COMPARE

Issues addressed by this pattern

FALSE PASS

4.1.62. SET CLEAR GOALS (Management Pattern)

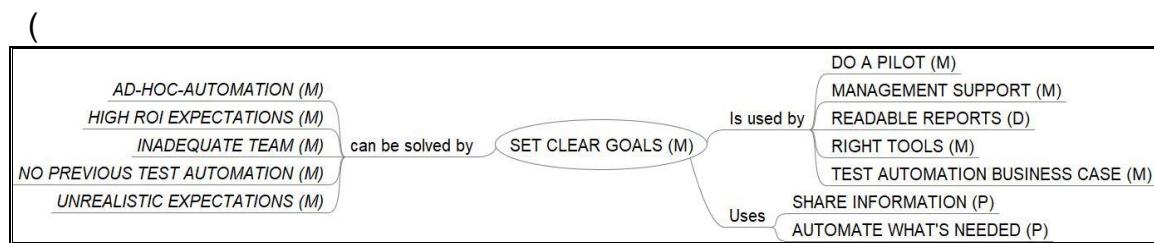


Figure 4.1.62-1 SET CLEAR GOALS

Pattern summary

Define the automation goals from the very beginning in a way that is clear and understandable to all.

Category

Management

Context

This pattern is always applicable, although the goals may be different at different stages. You need to know what your objectives and goals are when you first consider test automation, when an automation initiative is getting started, when your automation is going well, and when you want to re-vitalise a stalled automation effort.

If you don't know what your goals are, how do you know you are going in the right direction?

This pattern applies to the goals and objectives for the automated tests, but there should also be goals and objectives for monitoring the on-going health of the automation as a whole.

Also note that the goals for testing should be different to the goals for automation!

Description

If the automation objectives are defined clearly up front, we hope that people will not be disappointed later on because they expected different results.

Inform your managers about what is feasible and what is not. Show them magazine articles or papers that support your view. If you can, bring in an expert to clear up any misunderstandings.

Goals should be measurable so that you can tell if you have achieved them or not.

Implementation

Before you develop the goals, you should SHARE INFORMATION with management, testers and developers in order to understand what they need and what they are expecting from test automation. This should also be the time to inform management and testers about what test automation can deliver and what not.

Goals vary in different contexts and at different times, but here some suggested examples:

- Selected regression tests should run x-times faster and y-times more often.
- Tests too complex to perform manually are to be automated. Define exactly which ones.
- Support activities for manual tests that are to be automated.
- Increase the parts of the system that are tested in a release (coverage).
- Make it easy for business users to write and run automated tests.
- Ensure repeatability of automated regression tests.
- Free testers from repetitive and boring test running so they can spend more time doing exploratory testing.

Define from the beginning the scope of the automation: which tests are going to be automated and which are not, based on the identified goals. Be specific!

It may also be useful to consider the investment needed to achieve these goals, asking questions such as:

- What roles do you need on your test automation team?
- What return on investment do you expect?
- How high will the investments have to be? How much time and resources will you need?

Not good goals

There are some goals or objectives that seem attractive when you first encounter them, but they are not actually good goals for automation.

A common mistake is to confuse the goals for automation with goals for testing. This leads to goals like "Automated tests should find lots of bugs". This is fine for some types of automation, but not for regression test automation (which is what most people automate). Regression tests by their nature don't find many bugs, as they are running tests that passed the last time they were run and are checking to see that nothing has changed.

Another poor goal for automation is "Automate all of our manual tests". First, not all manual tests should be automated. For example, tests of some usability aspects need people to assess; it is a waste of time to automate a test that takes 2 weeks to automate, takes 1 hour to run manually, and is only needed once a quarter. We should only AUTOMATE WHAT'S NEEDED. But this erroneous goal also leads people to think that it is only manual tests that can be automated - but automation can do much more than just automate tests that can be run manually. For example, an automated test can check object states which are not visible to a human tester.

Example goals for tool selection: (See also RIGHT TOOLS)

The automated tests should be:

- Maintainable – to reduce the amount of test maintenance effort.

- Relevant – to provide clear traceability of the business value of automation.
 - Reusable – to provide modularity of test cases and function libraries.
 - Manageable – to provide effective test design, execution and traceability.
 - Accessible – to enable collaboration on concurrent design and development.
 - Robust – to provide object/event/error handling and recovery.
 - Portable – to be independent of the SUT and be completely scalable.
 - Reliable – to provide fault tolerance over a number of scalable test agents.
 - Diagnosable – to provide actionable defects for accelerated defect investigation.
 - Measurable – provide a testing dashboard along with customisable reporting.

Potential problems

Be sure to remind management that it may not be possible or advisable to automate every test case.

It may be difficult to get everyone to agree on what the goals should be, and particularly how they can be measured. Beware of inappropriate goals, such as "find lots of defects" when regression tests are automated.

Issues addressed by this pattern

AD-HOC AUTOMATION

HIGH ROI EXPECTATIONS

INADEQUATE TEAM

NO PREVIOUS TEST AUTOMATION

UNREALISTIC EXPECTATIONS

4.1.63. SET STANDARDS (Process Pattern)

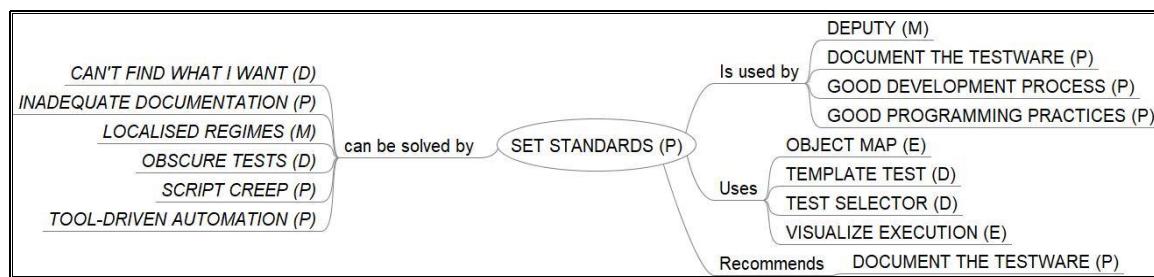


Figure 4.1.63-1 SET STANDARDS

Pattern summary

Set and follow standards for the automation artefacts.

Category

Process

Context

This pattern is appropriate for long-lasting automation. It is essential for larger organisations and for large-scale automation efforts. This pattern is not needed for single-use scripts.

Description

Set and follow standards: otherwise when many people work on the same project it can easily happen that everyone uses their own methods and processes. Team members do not “speak the same language” and hence cannot efficiently share their work; you get *OBSCURE TESTS* or *SCRIPT CREEP*.

As an extra bonus, standards make it easier for new team members to integrate into the team.

Implementation

Some suggestions for what you should set standards for:

Naming conventions:

- Suites: the names should convey what kind of test cases are contained in each test suite.
- Scripts: if the names are not consistent, an existing suitable script may not be found so a duplicate may be written.
- Keywords: it's important that the name immediately conveys the functionality implemented by the keyword.
- Data files: it should be possible to recognize from the name what the file is for and its status.
- If you implement OBJECT MAP, the right names facilitate understanding of the scripts and enable you to change tools without having to rewrite all of your automation scripts (only the tool-specific ones).
- Test data: if possible use the same names or IDs in all data files, as this will facilitate reuse.

Organisation of testware:

- Test Definition: Define a standard format or template to document all automated tests, for example as a standard block of comment, where the information is presented in a consistent way across all tests (e.g. same titles and levels of indentation). This should contain the following:
 - Test case ID or name.
 - What this test does
 - Materials used (scripts, data files etc).
 - Set-up instructions.
 - How it is called (input variables if any).
 - Execution instructions.
 - What it returns (including output variables).
 - Tear-down instructions.
 - Length (how long it takes to run).
 - Related tests.

- TEST SELECTOR tags.
- Any other useful information such as EMTE (Equivalent Manual Test Effort - how long this test would have taken manually).
- File and folder organisation and naming conventions

Other standards:

- Documentation conventions for scripts and batch files.
- Coding conventions for scripts.
- Data anonymization rules.
- Develop a TEMPLATE TEST.

Other advice:

- Document the standards!
- Standards should be reviewed periodically in order to adjust or enhance them.
- Put your standards in a Wiki so that everybody can access them at any time.
- If something has to be changeable, use a translation table so that the scripts can stay stable.
- Allow exceptions when needed.
- Setting standards for test data, for example using the test ID as the customer's name, can help to VISUALIZE EXECUTION as a way of monitoring test progress.

Potential problems

When devising standards, it is useful to get input from a selection of people who will be using the automation, to make sure that the conventions adopted will serve all of its users well. An additional benefit of getting others involved is that they will be more supportive of something that they helped to devise. Not many people like having standards imposed arbitrarily when they can't see the reason for them.

Once you have settled on your standards, however, then you need to make sure that everyone does use them in a consistent way, and this will take effort.

Issues addressed by this pattern

CAN'T FIND WHAT I WANT
INADEQUATE DOCUMENTATION
LOCALISED REGIMES
OBSCURE TESTS
SCRIPT CREEP
TOOL-DRIVEN AUTOMATION

4.1.64. SHARE INFORMATION (Process Pattern)

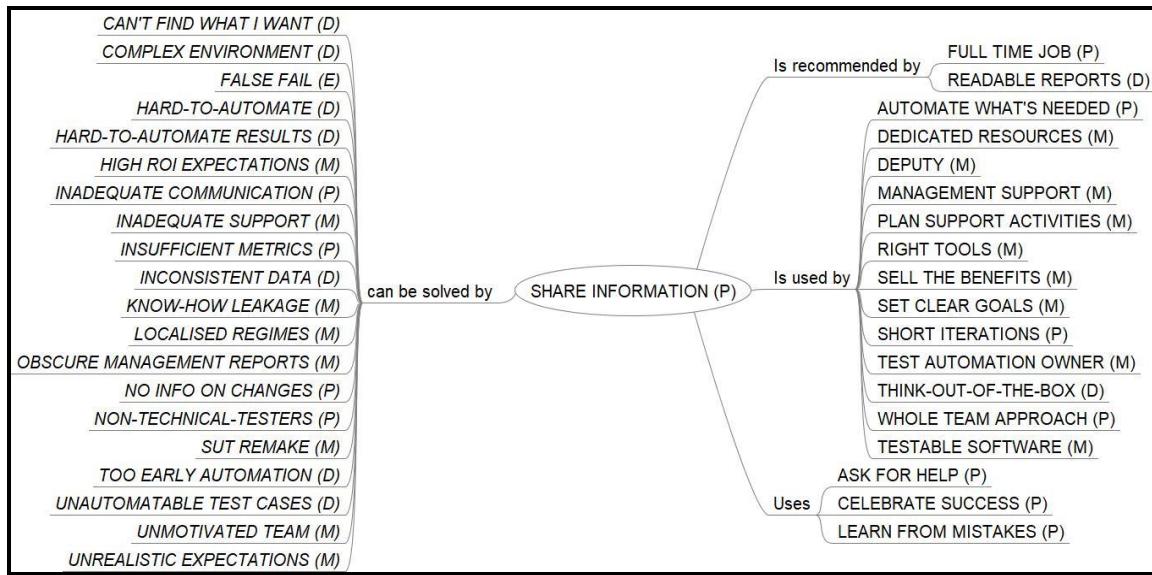


Figure 4.1.64-1 SHARE INFORMATION

Pattern summary

Ask for and give information to managers, developers, other testers and customers.

Category

Process

Context

This pattern is appropriate when you have to communicate with management, testers or developers, and when you have new people coming onto the team.

This pattern is not appropriate when you are working alone on issues that you have already mastered completely.

Description

There are many people who are involved with test automation, and they have different needs for what they need to know. But they won't know about things unless they are told, so you need to share relevant information with them at appropriate times.

Implementation

Some suggestions:

- Keep management informed about the progress of the test automation project. Find out what metrics they need, explain which can be easily collected and which not, and provide regular overviews in a format that is most appropriate for them.
- Have managers tell you what they specifically expect from test automation. In this way you can notice quickly if they have **UNREALISTIC EXPECTATIONS** and can inform them accordingly.

- Speak with other people about what you are doing: explaining something often leads to new ideas, yours or the people you are talking with.
- ASK FOR HELP when you have a problem or a question: you should never ponder too long on some issue, other people may have already solved just the same thing.
- Listen to testers or developers. Ask why they do something and why they do it as they do. If you find out what they really need, you can support them even better than you were planning.
- Ask developers to keep you informed when they make changes to the Software Under Test (SUT) that affect test automation.
- After you have obtained some concrete results, CELEBRATE SUCCESS
- Speak also about your failures: people will be thankful if in that way they can LEARN FROM MISTAKES.

Communication also includes reports, demonstrations, Wikis, notice boards etc. Use what is best known in your company.

Potential problems

Communication can easily be mis-interpreted, especially emails.

Communication needs to be at the right level for the recipient and tailored for the audience, or it will be ignored or worse.

Issues addressed by this pattern

CAN'T FIND WHAT I WANT

COMPLEX ENVIRONMENT

FALSE FAIL

HARD-TO-AUTOMATE

HIGH ROI EXPECTATIONS

INADEQUATE COMMUNICATION

INADEQUATE SUPPORT

INCONSISTENT DATA

INSUFFICIENT METRICS

KNOW-HOW LEAKAGE

LOCALISED REGIMES

NO INFO ON CHANGES

NON-TECHNICAL-TESTERS

OBSCURE MANAGEMENT REPORTS

SUT REMAKE

TOO EARLY AUTOMATION

UNAUTOMATABLE TEST CASES

HARD-TO-AUTOMATE RESULTS

UNMOTIVATED TEAM

UNREALISTIC EXPECTATIONS

4.1.65. SHARED SETUP (Design Pattern)

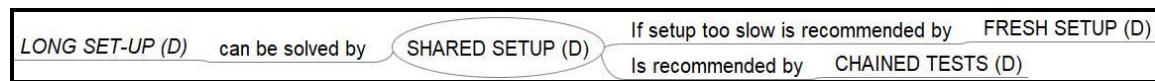


Figure 4.1.65-1 SHARED SETUP

Pattern summary

Data and other conditions are set for all tests before beginning the automated test suite.

Category

Design

Context

Use this pattern for long lasting and maintainable automation. By separating data required by each test within a common data set, it is possible to run the tests independently.

Description

Leave the Software Under Test (SUT) as it is after the test is run. In this way, after running the test, you can immediately check the state of the SUT, database contents etc. without having to restart the test and stopping it before it cleans up. The initial conditions (primarily data) are set for all tests before executing the automated suite. Data required by each test is already populated so no further setup is required. Tests don't clean up afterwards, so that if you want to check the results you can immediately control the status of the SUT.

Implementation

Initial conditions can be very diverse. Here some suggestions:

- Database configuration:
 1. Create the initial database data by adding independent data required by each test and padding as necessary.
 2. Avoid reusing data except for cases where tests are checking one another.
 3. Copy the initial database data once at the beginning of the test suite.
- File configuration:
 - Copy input or comparison files to a predefined standard directory.
- SUT:
 - each test should leave the SUT in the same state as the starting point for the next test case.

Potential problems

If the data set is dynamic (changing independently of the test), you should consider using instead a FRESH SETUP.

Issues addressed by this pattern

LONG SET-UP

4.1.66. SHORT ITERATIONS (Process Pattern)

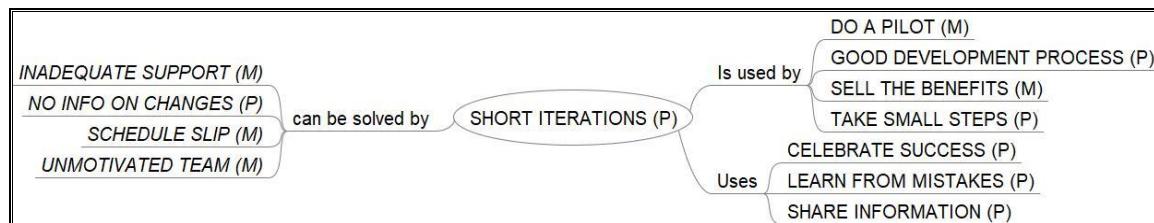


Figure 4.1.66-1 SHORT ITERATIONS

Pattern summary

Develop test automation in short iterations.

Category

Process

Context

This pattern is probably applicable for any context. Even if you are doing a large-scale long-term automation effort, it is worth using short iterations to develop it.

Description

Develop your test automation in short iterations. The charm of short iterations is that you get fast and regular feedback to see if you are on the right track. You can keep managers steadily informed about progress, testers and developers get to see results quickly, all of which increases the opportunities to get more support for the automation project.

At the end of the iteration your customers, who in the case of test automation are testers and developers, can immediately tell you if your efforts went in the right direction. If not, it's only the work of the past iteration that eventually gets scrapped and not the whole project!

Short iterations allow you to do automation that gives the best return at any point, so you are able to show benefits more quickly.

Implementation

As in Agile Software Development, you should first decide the iteration length (usually one or two weeks).

Then select for the iteration only as much work as can be completely finished, that is developed and tested.

At the end of the iteration (debriefing) you should examine what went well and what went wrong (LEARN FROM MISTAKES) in order to adjust the tasks in the next iterations. At this point you can also check to see what else could be improved, how well motivated the team are and address any problems.

Don't forget to SHARE INFORMATION and to CELEBRATE SUCCESS

Potential problems

It may be difficult to see what should go into an iteration - it wants to be something significant rather than trivial, but not too big to be done in a short iteration.

Working in short iterations is very intense; many iterations back-to-back may lead to burnout.

Thinking too much about the content for iterations may take your attention away from "the big picture".

The work done in an iteration needs to be able to be absorbed by those using the results.

Issues addressed by this pattern

INADEQUATE SUPPORT

NO INFO ON CHANGES

SCHEDULE SLIP

UNMOTIVATED TEAM

4.1.67. SIDE-BY-SIDE (Management Pattern)

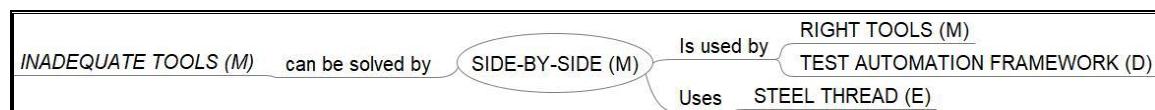


Figure 4.1.67-1 SIDE-BY-SIDE

Pattern summary

Determine which automation tool is best for you by trying them side by side.

Category

Management

Context

Use this pattern when you have to decide which tool or framework is best for testing your Software Under Test (SUT).

Description

If you have to select the tool that works best with the SUT and you have more than one candidate, the easiest way to choose is to try them out side by side. In this way it will be immediately apparent which tool serves your needs best.

Use this pattern also if you have to select a TEST AUTOMATION FRAMEWORK.

Implementation

Automate a STEEL THREAD with all the relevant tools and compare the results:

- Which tool can drive the SUT better?
- Which tool is easier to learn?
- Which tool is easier to use?

Issues addressed by this pattern

INADEQUATE TOOLS

4.1.68. SINGLE PAGE SCRIPTS (Design Pattern)



Figure 4.1.68-1 SINGLE PAGE SCRIPTS

Pattern summary

Develop an automation script for each window or page.

In the Selenium Community, this Pattern is known as "PAGE OBJECT".

Category

Design

Context

Use this pattern to build efficient, modular and maintainable testware.

It's not necessary for disposable scripts.

Description

In order to build modular scripts it is convenient to develop one for each window or page that will be driven by the tests.

A "page object" is a test object that holds the details of all the elements on a web page that might be involved in an automated test.

Implementation

If you use KEYWORD-DRIVEN TESTING you can implement this pattern by introducing a keyword for every window in your application. The keyword parameters will drive the GUI-Elements in the window.

For HTML-Pages map a UI page to a class, where for example a page is your HTML page. The functionality to interact or make assertions about that page is captured within the Page class. Then these methods may be called by a test.

Potential problems

This can become complex if it is not done well. The main benefit is increased maintainability of tests, so it is best used to reduce repetition of the same test code on multiple pages.

Issues addressed by this pattern

*GIANT SCRIPTS
OBSCURE TESTS*

4.1.69. SKIP VOID INPUTS (Execution Pattern)

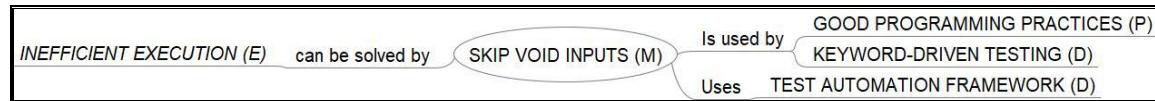


Figure 4.1.69-1 SKIP VOID INPUTS

Pattern summary

Arrange for an easy way to automatically skip void inputs

Category

Execution

Context

Use this pattern to write efficient reusable scripts. It's not necessary for disposable scripts.

Description

The more standardized your scripts the easier it will be to maintain them. To be able to use the same scripts even when not all data fields will be given, you should foresee some way to automatically skip void inputs

Implementation

Use a TEST AUTOMATION FRAMEWORK that supports skipping void inputs. See the examples in the wiki for some suggestions how to realize this

Issues addressed by this pattern

INEFFICIENT EXECUTION

4.1.70. SPECIFIC COMPARE (Design Pattern)

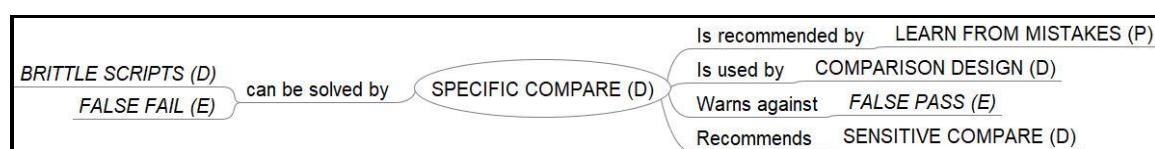


Figure 4.1.70-1 SPECIFIC COMPARE

Pattern summary

Expected results are specific to the test case so changes to objects not processed in the test case don't affect the test results.

Category

Design

Context

This pattern is applicable when your automated tests will be around for a long time, and/or when there are frequent changes to the SUT.

This pattern is not applicable for one-off or disposable scripts.

Description

The expected results check only that what has been performed in the test is correct. For example, if a test changes just two fields, only those fields are checked, not the rest of the window or screen containing them.

Implementation

Implementation depends strongly on what you are testing. Some ideas:

- Extract from a database only the data that is processed by the test case.
- When checking a log, delete first all entries that don't directly pertain to the test case.
- On the GUI check only the objects touched by the test case.

Potential problems

If all your test cases use this pattern you could miss important changes and get *FALSE PASS*. It makes sense to have at least some test cases using a *SENSITIVE COMPARE*.

Issues addressed by this pattern*BRITTLE SCRIPTS**FALSE FAIL*

4.1.71. STEEL THREAD (Execution Pattern)

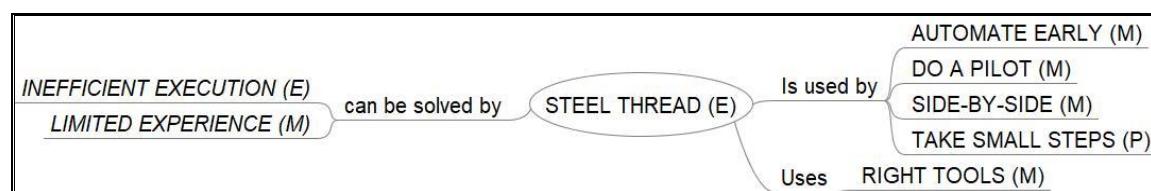


Figure 4.1.71-1 STEEL THREAD

Pattern summary

Test a thin slice of functionality that drives the Software Under Test (SUT) from one end to the other.

Category

Execution

Context

Use this pattern when you begin with test automation or when you start automating new functionality in the SUT that you first have to get to know.
This pattern is not necessary for writing disposable scripts.

Description

A steel thread is a test case that tests a “thin slice of functionality that cuts through the functionality from one end to the other” (Lisa Crispin). This kind of test cases is good to begin with because:

- You can support testers right away.
- It will also help you select the RIGHT TOOLS, since you will be able to get a good understanding of where the eventual automation problems are to be found.

Implementation

Select test cases that perform only minimal actions but go through the whole application. For instance, insert a new customer record with only the mandatory fields.

Issues addressed by this pattern

INEFFICIENT EXECUTION

LIMITED EXPERIENCE

4.1.72. TAKE SMALL STEPS (Process Pattern)

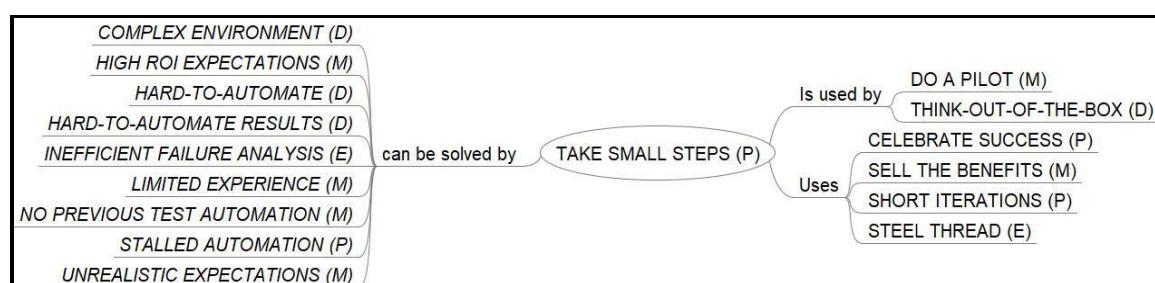


Figure 4.1.72-1 TAKE SMALL STEPS

Pattern summary

Look for small increments of the work that will give you value.

Category

Process

Context

This pattern is needed when you start with test automation and want to deliver results quickly. It is needed both for long lasting automation and for one-off or disposable scripts.

Description

Don't try to do too much too soon in your automation. It is tempting to try to automate as much as possible as quickly as possible, but this will ultimately make more work for you. Don't start with complex test cases, do first the easy and short ones. By taking small steps you can learn your way with tools, scripting etc. and you can show results relatively quickly so that you can SELL THE BENEFITS for your automation project.

Implementation

Here some suggestions:

- Automate at first only positive test cases (Happy Path).
- Automate only the smoke tests.
- Automate only bug fix tests.
- Automate stable functionality so that you don't have much maintenance in the beginning.
- Automate functionality for which a good test case base is already available.
- Automate the STEEL THREAD.
- Develop the automation in SHORT ITERATIONS
- CELEBRATE SUCCESS regularly.

In some cases, it can be most rewarding to do just some automation in order to support the testers right away. This will help to produce much interest and support for a later extension of the automation effort.

Potential problems

A step shouldn't be so small that it makes no difference, and if it is too large, then it isn't a small step.

Issues addressed by this pattern

*COMPLEX ENVIRONMENT
HARD-TO-AUTOMATE
HIGH ROI EXPECTATIONS
INEFFICIENT FAILURE ANALYSIS
LIMITED EXPERIENCE
NO PREVIOUS TEST AUTOMATION
STALLED AUTOMATION
HARD-TO-AUTOMATE RESULTS
UNREALISTIC EXPECTATIONS*

4.1.73. TEMPLATE TEST (Design Pattern)



Figure 4.1.73-1 TEMPLATE TEST

Pattern summary

Define a template test case as a standard from which you can drive all kinds of test case variations.

Category

Design

Context

Use this pattern to develop test automation efficiently. For disposable scripts it is not necessary but still quite useful.

Description

Define a standard automated test case. New test cases can be created by varying only one data field at a time. This makes it much easier to write new tests. In fact the process of varying the data may also be able to be automated (although remember that you also need to know what the expected result of the test will be).

Implementation

The standard test case should drive the Software Under Test (SUT) in what should be the most common usage, for instance you set up a standard user with the most common characteristics. To create new tests you can then, based on this benchmark, vary the details to check the possible combinations of values.

4.1.74. TEST AUTOMATION BUSINESS CASE (Management Pattern)



Figure 4.1.74-1 TEST AUTOMATION BUSINESS CASE

Pattern summary

Present a business case as for any other project, but provide also factors typical for test automation.

Category

Management

Context

You will need to apply this pattern if you have to convince management to support your test automation project.

It will not be necessary if you are only developing disposable scripts or if you already have good management support for automation without needing to make a business case. For example automation may be the only way to survive in agile development in a fast-moving industry.

Description

You present a business case as for any other project, but you should also provide factors typical for test automation, like the quality improvement due to broader testing or the reallocation of tester capacity to more challenging tasks once test automation has been implemented. These factors should be directly related to your own goals for automation.

Implementation

In order to succeed in test automation, you must first SET CLEAR GOALS: what do you want to achieve with the project? What not? It is crucial that you make very clear what test automation can or cannot deliver. This is important for everyone, but is particularly important when you make a business case for automation - if you set unachievable goals, you are guaranteed to fail!

The essence of a business case is to show that the benefits of the proposed course of action will be greater than its cost. The equation used for this is Return on Investment (ROI) = (benefit - cost) / cost. This gives a number (often quoted as a percentage). If the benefit is worth double the cost, then the ROI is 1 (or 100%). In order to compare manual testing to automated testing, first determine the costs of the manual testing process that you intend to replace with automation. Factor in the number of times the tests will (or should) be run. A given set of manual tests generally costs twice as much to run them 2 times, etc.

Then, estimate the planned costs for automation, including:

- Time and resources needed to select a suitable test tool.
- Tool licences and maintenance costs.
- Training.
- Learning times.
- Time and resources to develop the TESTWARE ARCHITECTURE.
- Time and resources to develop the test automation infrastructure.
- Maintenance costs for a test automation infrastructure.
- Time to build new automated tests.
- Maintenance costs for test automation testware.
- Time needed for refactoring the automation testware over time.
- Time to analyse failures found by the automated tests (this may be significantly greater than in manual testing.)

Now comes the more difficult part - estimating the benefits of automation, the return that could be expected on the investment. In order to use the ROI equation, the benefits need to be converted into the same units as the costs, usually money! Here are some ideas:

- Saved tester time = Time needed by the human tester to run a set of tests manually, minus the time needed for those same tests once they are automated. This is relatively easy to estimate using a "loaded salary cost" for the human testers.

- Number of automated test runs necessary for pay off - this can be calculated by the savings in human time multiplied by the number of times the tests would be run (or number of releases). When the savings catch up to the investment, then you are "in credit" and have a positive ROI.

But this is not the whole story. Other factors may not be easily converted into money to put into an ROI calculation but are an important part of a business case.

- The automation can free the testers from mundane and repetitive activities to concentrate on devising more and better tests.
- More tests can be run more often, so more of the software can be tested in each build or release. This increased coverage can give greater confidence that the system will work well.
- Improved morale of the testers, since the tools do the tedious bits and they can use their talents more fully to do better testing.

Don't forget to PLAN SUPPORT ACTIVITIES

Potential problems

The factors that may be easiest to put into a business case may not be all of the important factors, and may create a bias towards those factors. For example, it is easy to quantify that tester's time will be greatly reduced when tests are automated, compared to the time doing manual test execution. If this is the only factor taken into account, it may give the impression to managers that they won't need to have the testers any more! However, there are many other things that the testers need to do and there are activities that may take longer with automated tests than they did when tests were manual. Test automation should support testers, not replace them! A tool can never replace the human brain!

Issues addressed by this pattern

AD-HOC AUTOMATION

INADEQUATE SUPPORT

NO PREVIOUS TEST AUTOMATION

4.1.75. TEST AUTOMATION FRAMEWORK (Design Pattern)

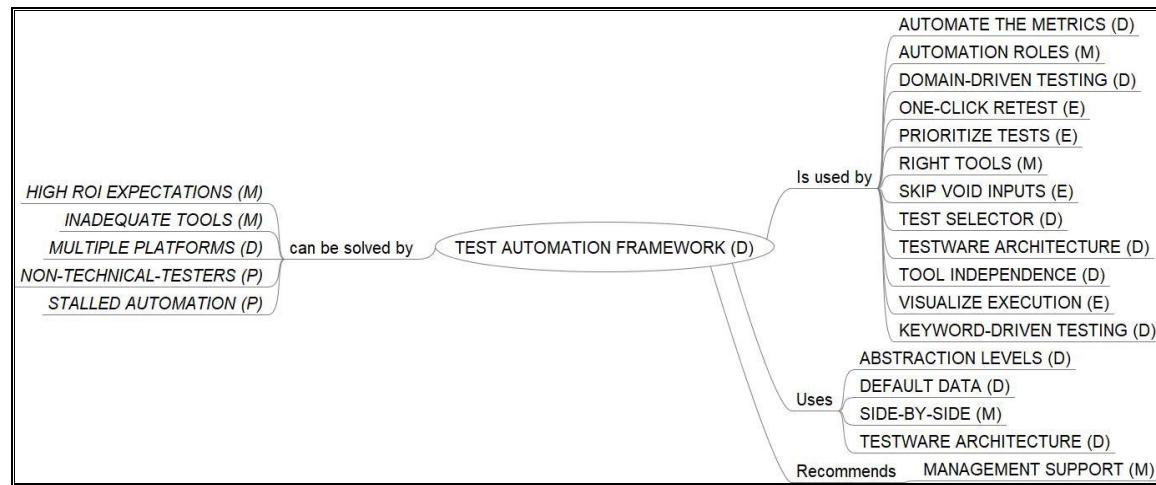


Figure 4.1.75-1 TEST AUTOMATION FRAMEWORK

Pattern summary

Use a test automation framework.

Category

Design

Context

This pattern is appropriate for long lasting automation.

If you just plan to write a few disposable scripts you will not need it.

Description

Using or building a test automation framework helps solve a number of technical problems in test automation. A framework is an implementation of at least part of a testware architecture.

Implementation

Test automation frameworks are included in many of the newer vendor tools. If your tools don't provide a support framework, you may have to implement one yourself.

Actually, it is often better to design your own TESTWARE ARCHITECTURE, rather than adopt the tool's way of organising things - this will tie you to that particular tool, and you may want your automated tests to be run one day using a different tool or on a different device or platform. If you design your own framework, you can keep the tool-specific things to a minimum, so when (not if) you need to change tools, or when the tool itself changes, you minimise the amount of work you need to do to get your tests up and running again.

The whole team, developers, testers, and automators, should come up with the requirements for the test automation framework, and choose by consensus. If you

are comparing two frameworks (or tools) use SIDE-BY-SIDE to find the best fit for your situation.

A test automation framework should offer at least some of the following features:

- Support ABSTRACTION LEVELS.
- Support use of DEFAULT DATA.
- Support writing tests.
- Compile usage information.
- Manage running the tests, including when tests don't complete normally.
- Report test results.

You will have to have MANAGEMENT SUPPORT to get the resources you will need, especially developer time if you have to implement the framework in-house.

Potential problems

It is not necessarily easy to acquire or make a good test automation framework, and it does take effort and time. But it is very worthwhile when done well. If you intend to develop a framework in-house make sure to plan for the necessary resources (developers, tools etc.) otherwise you could end up with a good framework that must be abandoned because for instance the only developer leaves the company (see issue *KNOW-HOW LEAKAGE* for more details).

Issues addressed by this pattern

HIGH ROI EXPECTATIONS

INADEQUATE TOOLS

MULTIPLE PLATFORMS

NON-TECHNICAL-TESTERS

STALLED AUTOMATION

4.1.76. TEST AUTOMATION OWNER (Management Pattern)

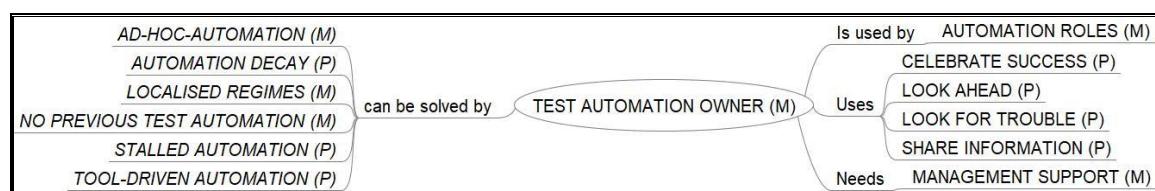


Figure 4.1.76-1 TEST AUTOMATION OWNER

Pattern summary

Appoint an owner for the test automation effort. If there is already a "champion" give him or her public support.

Category

Management

Context

This pattern is already necessary for individual automation efforts but is especially important for long lasting success by larger automation projects.

Description

The "test automation owner" is not necessarily the project leader, but he or she must be its "champion". The owner, once test automation has been established, controls that it stays "healthy" and keeps an eye on new tools, techniques or processes in order to improve it.

Implementation

The most important patterns for the automation owner are:

- LOOK AHEAD: keep in touch with the tester, automation and development community in order to stay informed about new tools, methods etc.
- LOOK FOR TROUBLE: watch out for possible problems in order to solve them before they become unmanageable.

Other useful patterns:

- CELEBRATE SUCCESS to keep the high motivation level in test automation.
- SHARE INFORMATION with testers, developers and management.
- MANAGEMENT SUPPORT will be needed at all times.

Issues addressed by this pattern

AD-HOC AUTOMATION

AUTOMATION DECAY

LOCALISED REGIMES

NO PREVIOUS TEST AUTOMATION

STALLED AUTOMATION

TOOL-DRIVEN AUTOMATION

4.1.77. TEST SELECTOR (Design Pattern)



Figure 4.1.77-1 TEST SELECTOR

Pattern summary

Implement your test cases so that you can turn on various selection criteria for whether or not you include a given test in an execution run.

Category

Design

Context

This pattern is needed when you have a lot of automated tests, when you can no longer run all of them in the allotted time.

Description

The need for this pattern is not obvious when you first start automating, but it is an important one to take into consideration if you want to have large-scale sustainable automation.

Implementation

As part of the description or documentation of a test, include some Tags to identify this particular test in different ways. For example, as part of a Smoke Test, a regression test for a particular feature, a depth test for a function, when the test last found a bug, even the test's author. Choose your tags depending on the different ways you might like to form subsets of tests for execution.

When you DOCUMENT THE TESTWARE, this is one of the things that you will SET STANDARDS for.

When you are gathering tests for an execution run, your TEST AUTOMATION FRAMEWORK should enable you to choose the tests to run by specifying the selector tags to include. For example, for tonight's overnight run, you may want to execute:

- The tests that failed last time they were run (to see if they are fixed correctly).
- The depth tests for the function that was most extensively changed.
- The highest priority tests from the full standard smoke test.
- Leslie's tests, as Leslie has been off sick for a few days.

Note that your framework needs to be designed to enable the tests to be selected by these Test Selectors!

The sets of tests that you choose will be determined by how you PRIORITIZE TESTS.

Potential problems

If this is not considered at the beginning of an automation effort, it is more difficult to implement, but is usually still worth doing, for the flexibility it gives to test execution.

The format of the Test Selector needs to be clearly defined and the standards for their definition and use enforced.

Issues addressed by this pattern

*INEFFICIENT EXECUTION
INFLEXIBLE AUTOMATION*

4.1.78. TEST THE TESTS (Process Pattern)

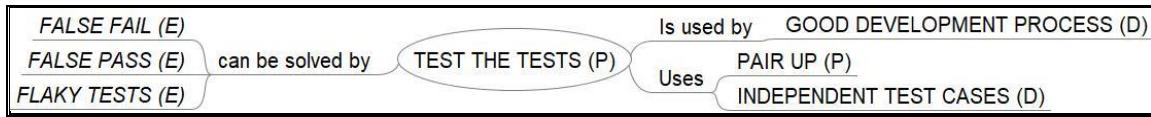


Figure 4.1.78-1 TEST THE TESTS

Pattern summary

Test the scripts just as you would test production code.

Category

Process

Context

This pattern is needed if you want to have reliable automation. (or if you don't believe in luck)

Description

Test your scripts individually, but also make sure that a failure in one test doesn't cause the following tests to fail too.

If you don't pay attention to testing your automated tests, you will probably end up spending a lot of time looking for bugs in the software being tested, which are actually bugs in your testware. You may well also have tests that pass when they should fail and vice versa, so your automation will become unreliable.

Implementation

New automation scripts should be tested just as other software.

Scripts can also be reviewed or Inspected before they are run and can be assessed using a static analysis tool (looking automatically for common types of script error). Automation scripts can also be tested by running them regularly and always checking the results. This is also the way to avoid *SCRIPT CREEP* because when the Software Under Test (SUT) changes the scripts can be updated or, if they no longer bring value, removed.

There are various actions to take in order to make your testware more robust:

- Implement INDEPENDENT TEST CASES, so that a test cannot fail just because a preceding one did.
- Make sure that when the tests run, they have all the resources they need (for instance enough memory, cpu etc) and that nothing else is using the same resources (databases, files etc).

Recommendations

If you regularly PAIR UP you can avoid many problems right from the beginning. As the saying goes: two pairs of eyes see better than one!

Potential problems

Testing the tests will take time, and you may find that you get into almost a recursive situation, where tests use other tests that should be tested, which in turn use other tests and where do you stop?

Issues addressed by this pattern

FALSE FAIL

FALSE PASS

FLAKY TESTS

4.1.79. TESTABLE SOFTWARE (Management Pattern)

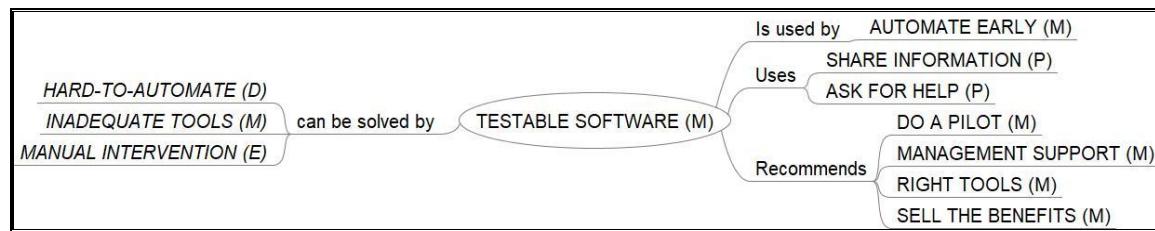


Figure 4.1.79-1 TESTABLE SOFTWARE

Pattern Summary

Locate what kind of implementations in the Software Under Test (SUT) may make test automation difficult or impossible, and find a solution as early as possible.

Category

Management

Context

Use this pattern when you start test automation from scratch.

You will not need it to just write disposable scripts.

Description

Find out as soon and as fast as possible what could be a problem in order to make sure that the SUT will support automated testing from the very beginning. Both for a new project and for an older application, you should ASK FOR HELP from the developers. When the problem is understood, it is usually not difficult to find a work-around that works for both testers and developers.

If something is impossible to automate, or not economically worth the effort to automate it, make sure that these are tested manually, and try to see if it might be possible to write the software in a way that could be tested automatically in the future.

Implementation

SHARE INFORMATION with the developers to discuss the problem with them:

- Find out what kind of components they are using. Contact the tool vendor to ask for support.
- If the results keep changing, examine how you are recording them. If you check them directly in the GUI the results will be much more sensitive to eventual changes in the SUT than if you check them by extracting the expected data from the database tables.
- If the time waiting for responses from the SUT is variable, build synchronization points in your scripts. For instance have the script wait until the hourglass disappears or a button is enabled.
- Sometimes small adjustments in the code of the SUT can solve the problem:
 - Make sure that every object you need is uniquely named.
- When some computation takes time, ask the developers to mark in some way when it is finished. For instance a disabled button will be enabled at the end

Recommendations

DO A PILOT to find out what kind of implementations may cause trouble. Changes to the SUT or in the way it is implemented may significantly affect the automation. Developers should be made aware of this, and MANAGEMENT SUPPORT may be needed to ensure the right balance between essential changes and ensuring that the software is testable and automatable.

SELL THE BENEFITS:

- show management which returns would be possible at what costs.
- show developers the advantages of getting timely feed-back through test automation.

List the “Do NOTs” in a Wiki so that developers can check which programming practices or which components do not support test automation.

Try to get at least some automation running as fast as possible. As soon as the developers start getting good feedback from the automated tests, they will be much more willing to help support it.

Potential problems

If you ascertain that the tool just doesn't fit your application, then you should select another one (RIGHT TOOLS)

Issues addressed by this pattern

HARD-TO-AUTOMATE

MANUAL INTERVENTIONS

INADEQUATE TOOLS

4.1.80. TESTWARE ARCHITECTURE (Design Pattern)

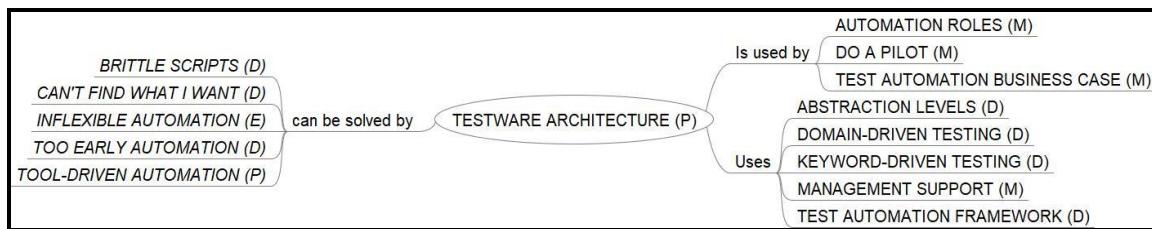


Figure 4.1.80-1 TESTWARE ARCHITECTURE

Pattern summary

Design the structure of your testware so that your automators and testers can work as efficiently as possible.

Category

Design

Context

This pattern is particularly important for large-scale or long-lasting automation. If you just plan to write a few disposable scripts you will not need it.

Description

There are many artefacts in test automation and all of them need to have a place to "live". Scripts, data sets, expected results, actual results, utilities etc. For efficient working, it is important to ensure that testers and automators know where to find artefacts and can access them quickly and easily as they work. For example, a standard structure for testware permits the information of where artefacts are to be found or placed to be made known to the tools, for example, when tests are to be run, or test results are reported. This makes the job of interacting with the tools simpler and less error-prone, because the "where to find it" information is already built into your tools or framework.

Whichever tool you use will have some kind of architecture as the default for that tool. One option is to simply adopt the tool's architecture. Although this may be the easiest option in the beginning, it could have long-term negative consequences, as you will then be "tied" to that tool's current architecture.

It is much better to design the architecture of your tests so that it supports the way you want to use your automation. It is relatively straight-forward to adapt the testware to the needs of the tool (e.g. copy a script to the place where the tool expects to find it) - and this can and should also be automated.

Implementation

When you design your architecture, you need to decide where you will store various testware artefacts, i.e. what filing structure will you use, what naming conventions will you use for files and folders, etc.

It is important to distinguish between the test materials and the test results. The test materials are those artefacts that should be in place before a test is executed,

such as the test inputs, expected results, any set-up that needs to be done, required data before a test runs, environment settings, and the description / documentation for that test. Many of these artefacts will be used by more than one test - in this case, they should be accessible to all tests that use them, but there should only be one master copy (except for legitimate different versions). These artefacts must be under configuration control to prevent people over-writing edits, for example.

The test results include everything that is produced by the system or software when a test is executed, including the actual results, log files, difference files (between actual and expected results), etc. There are usually many sets of these for a given test, at least for a test that is run many times. These need to be stored in a different way to the test materials, since one set of materials (for a single test) will produce a new copy of the test results each time it is run. If tests are run daily or more often, these sets of results will soon build up, and you don't want them "clogging" the storage of your tests.

It is worth asking whether you need to keep all of the test results each time - you may want to keep the test log, to prove that the test was run, but if the test passes, then the actual results has exactly matched the expected results, so why keep two copies of the same thing?

Your testware architecture should implement ABSTRACTION LEVELS. Make sure that you design the structure of your scripts so that tool-specific scripts are kept to a minimum. To make your automation accessible to a wide variety of testers, including those who are not programmers, make sure that you enable them to write and run automated tests easily. DOMAIN-DRIVEN TESTING and KEYWORD-DRIVEN TESTING are good ways to implement this.

You will have to have MANAGEMENT SUPPORT to get the time you need to design a long-lasting effective architecture for your automated tests.

A TEST AUTOMATION FRAMEWORK is an implementation of at least part of a testware architecture.

Potential problems

It is not necessarily easy to design a good testware architecture, and it does take effort and time. But it is very worthwhile when done well.

Issues addressed by this pattern

BRITTLE SCRIPTS

INFLEXIBLE AUTOMATION

CAN'T FIND WHAT I WANT

TOO EARLY AUTOMATION

TOOL-DRIVEN AUTOMATION

4.1.81. THINK OUT-OF-THE-BOX (Design Pattern)

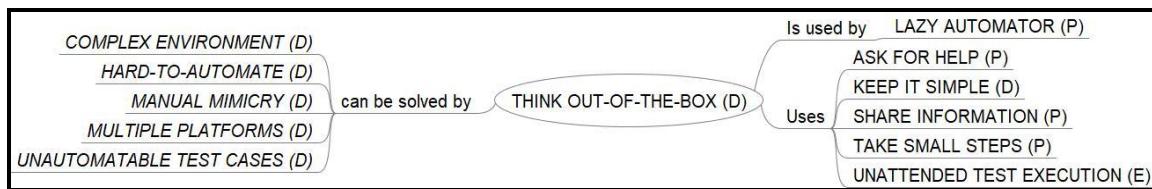


Figure 4.1.81-1 THINK OUT-OF-THE-BOX

Pattern summary

The best automation solutions are often found by concentrating on what the test case is trying to check and forgetting how it is executed manually.

Category

Design

Context

This pattern is always valid.

Description

When tackling some automation problem, it pays to look for unconventional solutions.

Implementation

There are quite a few ways to implement this pattern. Here some ideas:

- For a start KEEP IT SIMPLE and TAKE SMALL STEPS. Once you better understand the problem try to think about it from different viewpoints.
- Can you automate some of the tasks surrounding the test automation? For example, as part of pre-processing for a given set of tests, perhaps some data needs to be set up in a database or file. Write (or ask someone to write for you) a small utility or script to populate the data you need. This can then be called as part of the set-up for that test, and this helps toward UNATTENDED TEST EXECUTION.
- ASK FOR HELP: ask a tester about different ways to get the same results or ask a developer how the functionality you want to test is implemented.
- SHARE INFORMATION: Explain to testers, developers or other automators your problem, maybe all together you come on a better solution.
- Search in internet forums how other people have solved the same problem.
- Sleep over it: sometimes in the evening a problem hovers like an impregnable wall, but in the morning suddenly the wall has become much lower!

Potential problems

If you don't get a better idea immediately, use what you have: a not so good solution is still much better than no solution at all

Issues addressed by this pattern

*COMPLEX ENVIRONMENT
HARD-TO-AUTOMATE
MANUAL MIMICRY
MULTIPLE PLATFORMS
UNAUTOMATABLE TEST CASES*

4.1.82. TOOL INDEPENDENCE (Design Pattern)

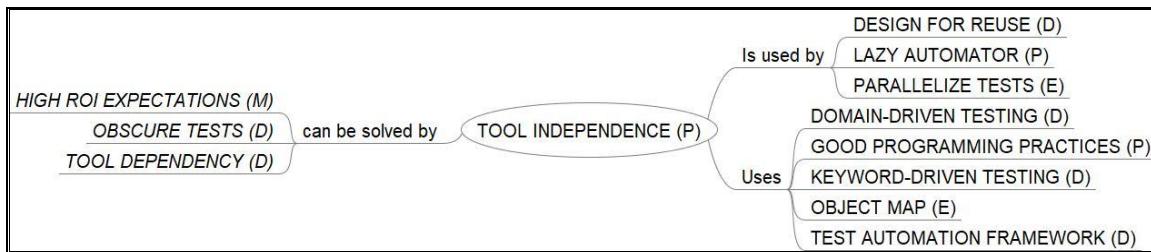


Figure 4.1.82-1 TOOL INDEPENDENCE

Pattern summary

Separate the technical implementation that is specific for the tool from the functional implementation of the tests.

Category

Design

Context

This pattern is applicable if you want to run automated tests on multiple platforms or environments, or if the tool you are using might change at some point (which is more likely the longer you have automated tests!).

This pattern is not applicable for short-term automation, e.g. disposable scripts.

Description

Design the structure for the testware so that tool-specific elements are kept to a minimum. Make the scripts modular, where tool-specific scripts are called by the scripts implementing the tests.

Implementation

Some suggestions:

- Use a TEST AUTOMATION FRAMEWORK that supports KEYWORD-DRIVEN TESTING or DOMAIN-DRIVEN TESTING. Having separated the functional scripts from the tool-dependent ones means that if you change the tool, you only have to rewrite the tool-specific command scripts and all the others can be used without change.
- Use an OBJECT MAP to name the GUI elements in your application. If you change your tool, you will have to map the GUI elements again in the new tool, but you will not need to change the functional scripts.

- Use GOOD PROGRAMMING PRACTICES to keep tool-specific aspects in a small number of scripts that can be called by other scripts.

Potential problems

The effort spent on making testware separate from tool-specific aspects may be considered a waste of time by those who cannot see that the tool "engine" will change in the future.

Issues addressed by this pattern

HIGH ROI EXPECTATIONS

OBSCURE TESTS

TOOL DEPENDENCY

4.1.83. UNATTENDED TEST EXECUTION (Execution Pattern)

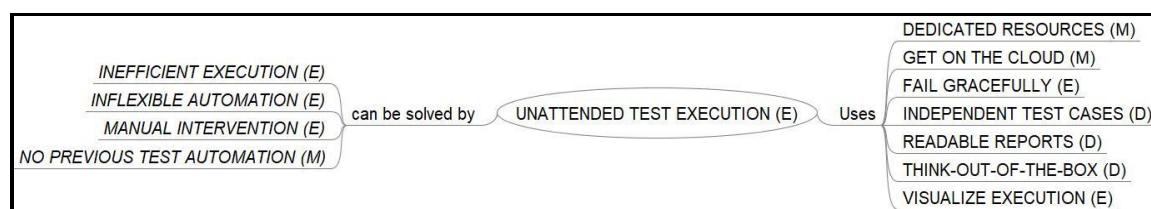


Figure 4.1.83-1 UNATTENDED TEST EXECUTION

Pattern summary

Automated tests should start automatically and run unattended.

Category

Execution

Context

This pattern is appropriate when your automated tests will be around for a long time.

This pattern is not appropriate for one-off or disposable scripts.

Description

Automated tests bring the most Return on Investment when they are scheduled to start automatically and run unattended. Results should be presented so that testers only have to check for failures (for instance colour coded).

Implementation

In order to support unattended test execution, you must set up a supporting infrastructure. Some suggestions:

- DEDICATED RESOURCES: run your tests on dedicated machines to avoid disruptions by other users.

- GET ON THE CLOUD: in the cloud you can easily (and cheaply) simulate as many environments as you need.
- Set up the “Planned Task” feature to start your tests automatically at a given date and time.
- Design INDEPENDENT TEST CASES so that tests don’t disrupt each other.
- FAIL GRACEFULLY so that failed tests don’t compromise a whole test suite.
- VISUALIZE EXECUTION so that you can peek in at any time to find out how far the execution has got.
- Produce READABLE REPORTS that enable testers to see at a glance if and where there has been a failure.
- THINK OUT-OF-THE-BOX to find tasks that can be automated around the test execution, for example in pre-processing (set-up) or post-processing (analytical tools for test results).

Potential problems

People looking for whether the tests passed (green) or failed (red) may be colour-blind! (Yes, it has happened.)

The tests need to be able to recover from catastrophic failures, and this does take effort to build in.

Issues addressed by this pattern

*INEFFICIENT EXECUTION
INFLEXIBLE AUTOMATION
MANUAL INTERVENTIONS
NO PREVIOUS TEST AUTOMATION*

4.1.84. VARIABLE DELAYS (Execution Pattern)

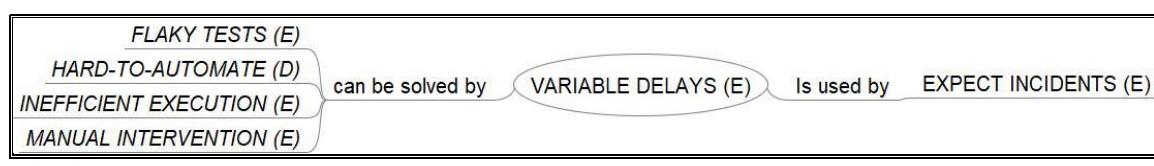


Figure 4.1.84-1 VARIABLE DELAYS

Pattern summary

Use variable delays based on events, not fixed delays based on elapsed time.

Category

Execution

Context

Use this pattern to make your test run as fast and as efficiently as possible.

Description

When SUT execution is slow it may be necessary to add delays to the automation script. Fixed length delays should be avoided, because in order to account for different test environments, traffic situations and so on you have to use the longest possible pause. New generation tools offer the possibility to wait only until an event happens. The tool will wait the given maximum wait-time only if the event doesn't occur at all.

Implementation

The actual implementation depends not only on the tool you are using, but also on how the System under Test (SUT) has been developed. Some suggestions when developing variable delays:

If the SUT shows when an action has been terminated, check it and delay execution until it switches auf true:

- a window disappears or appears.
- a GUI-Component is enabled or disabled.
- a database field is updated.

With files, folders or databases there are usually parameters that tell the system to wait until the action is done

Issues addressed by this pattern

FLAKY TESTS

HARD-TO-AUTOMATE

INEFFICIENT EXECUTION

MANUAL INTERVENTIONS

4.1.85. VERIFY-ACT-VERIFY (Design Pattern)



Figure 4.1.85-1 VERIFY-ACT-VERIFY

Pattern summary

The action to test is surrounded by two verifications that check the initial and final state (pre- and post-conditions).

This pattern ensures that a test actually checks whether a function works as expected, and a failure isn't due to some other factor (e.g. data consistency during creation / update or deletion of objects).

Category

Design

Context

This pattern is especially useful for long term automation, but it can be used also for short lived tests.

Note that if you do a FRESH SETUP for every test case you will not really need this pattern since you supposedly set up consistent initial conditions

Description

Using this pattern lets you ensure that the function you are testing actually works in the desired way. The pattern reminds you to check that the tested action triggers the right state transition. A test without verifications may not detect a problem if e.g. the steps to perform an action were successful but the action itself wasn't (e.g. the individual actions to create a customer were successful, but the saving to the database failed).

Implementation

The action to test in the test case is surrounded by two verifications that check the initial and the final state (verify-act-verify).

The advantages of using this pattern are:

- Higher detection rate of defects in functions.
- Well-structured test cases --> Tester cannot forget a verification.
- Easy to use.

Examples:

- Create a new user:
 - Verify that the user does not already exist.
 - Create the user.
 - Verify that the new user exists.
- Update an existing user account:
 - Verify that the user does not already exist.
 - Create the user.
 - Verify that the new user exists.
- Delete a user:
 - Verify that the user already exists.
 - Delete the user.
 - Verify that the user is deleted.

Potential problems

- May not applicable for every type of use case (e.g. testing search functions may not be suitable for VAV).
- May increase the size and execution length of tests that require relatively complex verifications for simple actions.

Issues addressed by this pattern

FALSE PASS

FLAKY TESTS

4.1.86. VISUALIZE EXECUTION (Execution Pattern)

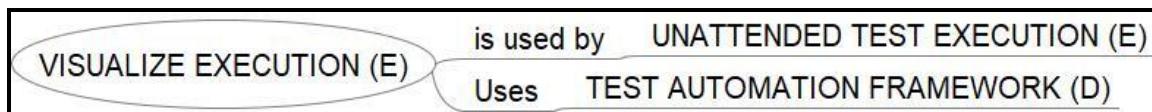


Figure 4.1.86-1 VISUALIZE EXECUTION

Pattern Summary

When running tests display which test case is currently executing.

Category

Execution

Context

This pattern is helpful when running unattended tests.

Description

When executing long-running automated tests it is quite useful to be able to see at a glance which test case is currently executing. To achieve this you should display the test case Id somewhere on the screen.

Implementation

Some suggestions:

- If you are testing windows in which you have to insert text that is inconsequential for the test case, simply insert the test case id.
- If your test tool or TEST AUTOMATION FRAMEWORK supports it, write the test case information to some otherwise unused screen area.

4.1.87. WHOLE TEAM APPROACH (Process Pattern)

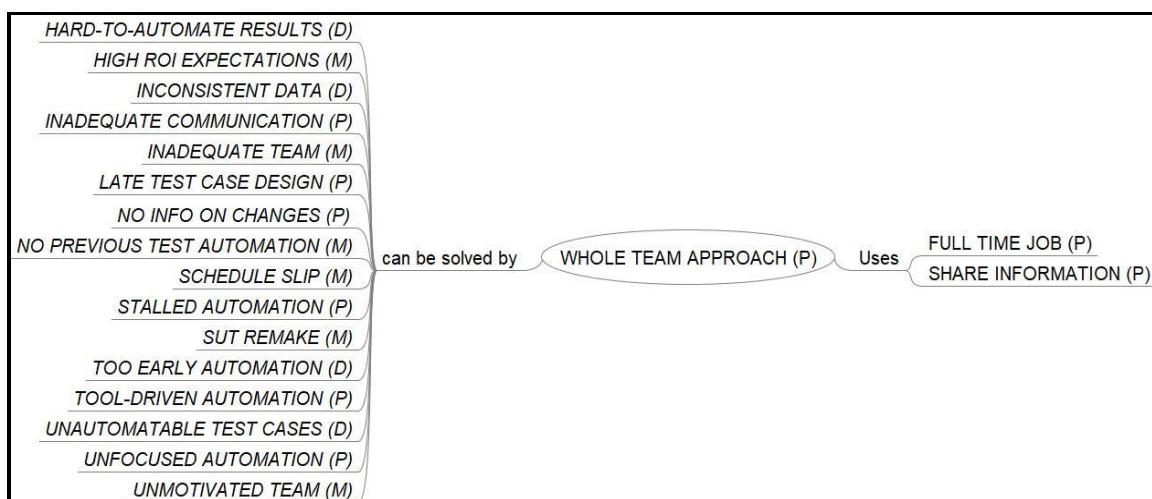


Figure 4.1.87-1 WHOLE TEAM APPROACH

Pattern summary

Testers, coders and other roles work together on one team to develop test automation along with production code.

Category

Process

Context

This pattern is most appropriate in agile development but is effective in many other contexts as well. This pattern is not appropriate if your team consists of just you.

Description

Everyone collaborates to do test automation. Developers build unit test automation along with production code. Testers know what tests to specify, automators or coders help to write maintainable automated tests. Other roles on the team also contribute, eg, DBAs, system administrators.

Implementation

If you are doing agile development, you should already have a whole-team approach in place for software development, testing and test automation.

If you are not doing agile, it is still very helpful to get a team together from a number of disciplines to work on the automation. In this way you will get the benefit of a wider pool of knowledge (SHARE INFORMATION) which will make the automation better, and you will also have people from different areas of the organisation who understand the automation.

Potential problems

If people are not working on the automation as a FULL TIME JOB, there may be problems as other priorities may take their time away from the automation effort. Another possible problem occurs when many DevOps teams develop test automation independently (look up the issue *LOCALISED REGIMES*)

Issues addressed by this pattern

*HARD-TO-AUTOMATE RESULTS
HIGH ROI EXPECTATIONS
INADEQUATE COMMUNICATION
INADEQUATE TEAM
INCONSISTENT DATA
LATE TEST CASE DESIGN
NO INFO ON CHANGES
NO PREVIOUS TEST AUTOMATION
SCHEDULE SLIP
STALLED AUTOMATION
SUT REMAKE
TOO EARLY AUTOMATION
TOOL-DRIVEN AUTOMATION
UNAUTOMATABLE TEST CASES
UNFOCUSSED AUTOMATION
UNMOTIVATED TEAM*

5. Appendix

5.1. Recipes

5.1.1. Tiramisu

Prepare the day before!

Ingredients:

- 4 eggs
- 4 tablespoons of sugar
- Amaretto liquor
- 500g Mascarpone
- 1-2 big packages of sponge fingers
- very strong coffee
- cocoa powder

Cream:

1. Separate the eggs.
2. Whip egg yolk with the sugar (in a big bowl) until it forms a solid white cream.
This takes a while.
3. Whisk egg white until stiff.
4. Add mascarpone and egg white to the egg yolk and carefully mix until it forms a smooth cream.
5. Add a shot of Amaretto.

Preparation:

1. Fill a cup (or a glass) about 3-4 cm high with coffee.
2. In a second cup add about 2cm of amaretto.
3. Take the sponge fingers and dip one end in the coffee and the other end into the Amaretto and cover the bottom of a casserole. Whenever necessary refill the cups.
4. Once the bottom of the casserole is covered, spread the cream over the dipped sponge fingers, so that they are almost not visible anymore.
5. Add more layers of sponge fingers and cream until both are used up.
6. The top layer should be cream and contain somewhat more cream than the other layers.
7. Sift the cocoa powder onto the cream until the cream cannot be seen anymore.
8. Leave to rest overnight, ideally in a cool place. Enjoy the day after. Store in the fridge and consume within 3 days.

5.1.2. Lentil soup

Ingredients:

- 1 large onion, peeled and chopped
- 2 - 4 carrots, peeled and sliced thin

- 4 rashers of back bacon, finely chopped (omit for vegetarian version)
- 4 cloves of garlic, peeled and chopped fine
- 4 oz / 100g butter
- 1 Tablespoon vegetable oil
- 1 lb / 500g red lentils
- 4 ½ pints / 2 litres ham stock (or vegetable stock for vegetarian version)
- 1 pint / 450 ml milk
- salt and pepper to taste

Preparation:

1. Heat the butter and oil in a large deep pan and add the onion, carrots, bacon and garlic. Sweat them gently, stirring occasionally, for 5 to 10 minutes until the veg is softened but not coloured.
2. Add the lentils, stir in thoroughly and sauté until they appear translucent and shiny.
3. Add the stock to the pan, stirring well, and turn up the heat. Bring the pan to the boil, then reduce the heat and simmer until the vegetables are tender, stirring occasionally to prevent the lentils sticking to the bottom of the pan (this should take 20 to 30 minutes).
4. Add the milk, season to taste and then liquidise the soup using a stick blender, blender or food processor. Bring back to a gentle simmer, check the seasoning again and serve.

5.1.3. Eggplant Parmigiana

Ingredients (Serves 4):

- 4 small eggplants or two big ones
- Butter
- Bolognese Sauce
- Grated Parmesan
- Mozzarella or other cheese slices

Preparation:

1. Cut the eggplants in thin slices and grill them in the oven for about 5 minutes (they must get only slightly brown).
2. Butter a baking dish.
3. Lay a layer of eggplant slices in the baking dish.
4. Cover with a layer of Bolognese sauce, grated Parmesan and cheese slices.
5. Repeat 3 and 4 until you are through with the eggplant slices.
6. Bake until the cheese melts and the eggplants are soft and juicy (at least an hour).
7. Serve.

5.1.4. Bolognese Sauce

Ingredients

- 2 cans peeled tomatoes
- 1 small onion
- 1 carrot
- A bunch of parsley
- 1 celery stalk (optional)
- 1 big spoonful butter
- Olive oil
- 0,5 kg minced beef (or more)
- Water
- Salt
- Pepper

Preparations:

1. Chop the onion, the carrot, the parsley (and the celery).
2. Heat the butter with enough oil to cover the bottom of the pan.
3. Hot broil the onion, carrot, parsley (and celery).
4. After about 5 min. add the meat and broil until nice and brown (not burned!).
5. Add the tomatoes. While they cook break them up with a fork.
6. Add salt and pepper.
7. When the liquid has almost dried up add a glass of water and continue cooking at high heat. You must be careful that it doesn't burn.
8. Repeat No. 7 at least two more times. You should get a thick sauce.

5.1.5. Liz's special salad

Ingredients (serves 4):

- 1 lettuce (you can also mix other kinds of salad, like romaine lettuce, arugula, lambs lettuce etc.)
- 10 cherry tomatoes (any tomato that tastes like a tomato!)
- 1 sweet red pepper
- 1 clove garlic
- 2-4 soup spoons olive oil
- Balsamico Vinegar
- Salt

Preparations:

1. Pour 1 cap of vinegar in the salad bowl.
2. Add salt until it stops melting in the vinegar.
3. Add very thin slices of garlic.
4. With a spoon break the garlic by pressing it in the vinegar.

5. Add the oil and mix thoroughly.
6. Cut up the tomatoes, add them to the sauce and mix again.
7. Add the pepper cut in small pieces and mix well.
8. Wash and dry the salad. If needed break up the leaves and add.
9. Mix well a couple of times. Everything must be wet with the sauce.
10. Serve.

5.1.6. Toffee sauce (topping for ice cream)

Ingredients (serves 4):

- 30 g butter
- 90 g light brown sugar
- 30 g cream or milk

Preparations:

1. Melt the butter in a small pan.
2. Add the brown sugar and stir until it is all dissolved.
3. Remove from heat.
4. Add cream or milk until it is pouring consistency.

5.1.7. Chicken Hungarian style

Ingredients:

1. 1 lb / 500g Chicken wings cut apart
2. 1 big onion chopped
3. 1 level tbsp. sweet red paprika powder
4. Olive oil
5. 1 tub sour cream
6. 0,83lb / 375g Flour
7. 2 eggs
8. Water
9. Salt

Preparations:

Chicken:

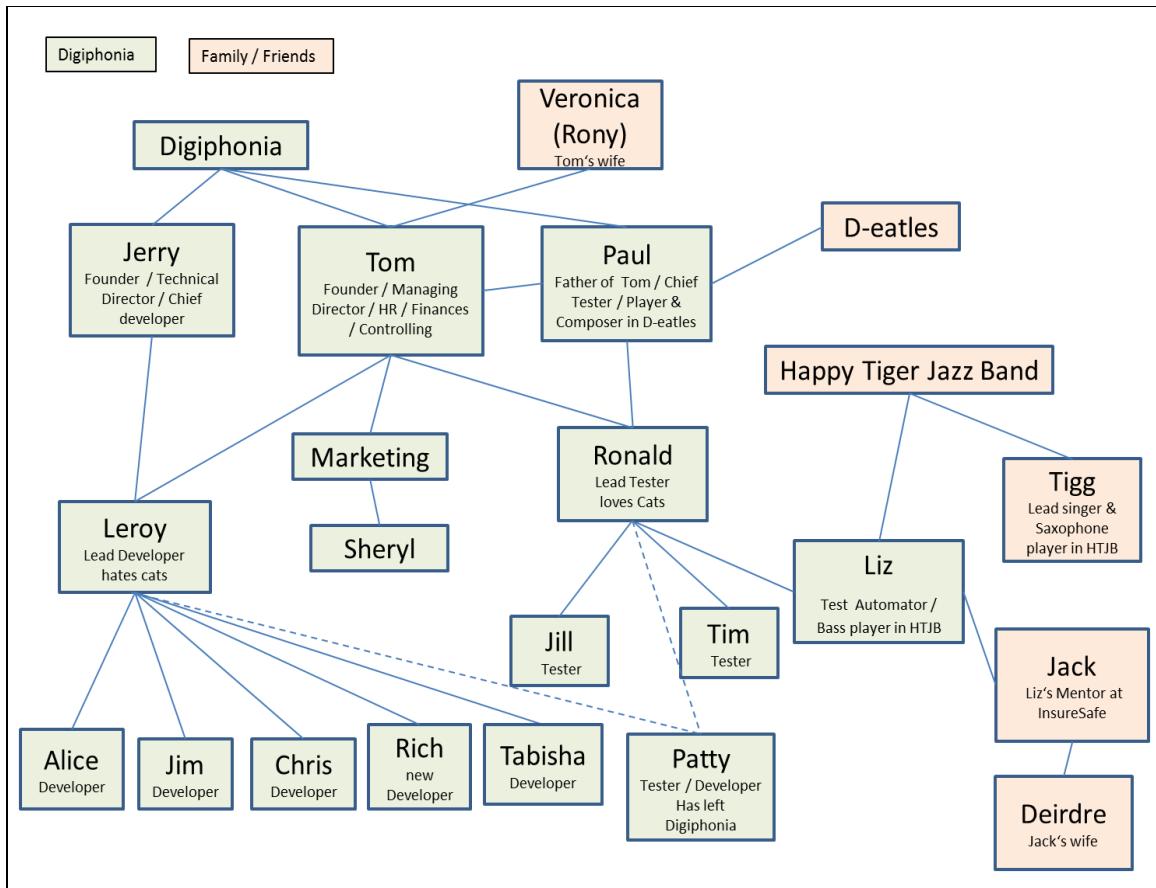
1. Heat the onion in the oil until translucent.
2. Salt the chicken pieces and add them to the onion.
3. Add the paprika powder and mix constantly while broiling at a lower temperature. Be very careful not to burn the paprika (burned paprika tastes awful!).
4. When the chicken has gotten some colour, place a lid on the pan and let simmer at low temperature for about 15-20 min.

5. Add the sour cream and serve together with the nockerli.

Nockerli:

1. Mix flour, eggs and salt to a soft and sticky dough. Eventually add water.
2. Prepare a pot with salted boiling water
3. Put the dough on a cutting board and cut small pieces directly into the boiling water
4. When all the nockerli have swam up, pour them out and serve

5.2. Overview of relationships between story characters



5.3. Index

- ABSTRACTION LEVELS 13, 14, 22, 30, 31, 32, 33, 34, 36, 62, 81, 83, 86, 88, 90, 91, 92, 102, 107, 148, 157, 180, 182, 189, 200, 201, 203, 204, 211, 239, 240, 253, 269, 313, 320
definition 211
hamburger model 31
- AD-HOC AUTOMATION 9, 11, 77, 230, 237, 272, 295, 311, 314
definition 145
- ASK FOR HELP 60, 96, 125, 147, 183, 213, 220, 250, 262, 278, 280, 299, 318, 322
definition 213
- AUTOMATE EARLY 161, 214
definition 214
- AUTOMATE GOOD TESTS 115, 161, 203, 215, 218
definition 215
- AUTOMATE THE METRICS 176, 216
definition 216
- AUTOMATE WHAT'S NEEDED 114, 115, 192, 207, 214, 217, 273
definition 217
- AUTOMATE WHAT'S NEEDED 182, 295
- AUTOMATION DECAY 214, 265, 266, 272, 286, 314
definition 146
- AUTOMATION ROLES 16, 167, 173, 182, 191, 219, 235
definition 219
- BRITTLE SCRIPTS 29, 30, 76, 80, 83, 213, 221, 228, 254, 258, 269, 272, 274, 306, 321
definition 147
- BUGGY SCRIPTS 252, 254
definition 149
- CAN'T FIND WHAT I WANT
definition 149
- CAN'T FIND WHAT I WANT 237, 298, 300, 321
- CAPTURE-REPLAY 47, 67, 157, 204, 220, 227
definition 220
- CELEBRATE SUCCESS 52, 120, 127, 182, 208, 221, 291, 299, 302, 308, 314
definition 221
- CHAINED TESTS 40, 185, 223, 255
definition 223
- CHECK-TO-LEARN 183, 223, 250
definition 223
- COMPARE WITH PREVIOUS VERSION 160, 173, 225
definition 225
- COMPARISON DESIGN 30, 81, 84, 148, 154, 155, 172, 180, 226
definition 225
- COMPLEX ENVIRONMENT 237, 256, 300, 308, 322
definition 150
- DATA CREEP 231, 238, 252, 254, 256, 259, 264, 267, 269, 272, 286
definition 151
- DATA-DRIVEN TESTING 17, 22, 33, 47, 48, 68, 71, 72, 88, 89, 90, 196, 212, 214, 227, 236, 241, 253
definition 227
- DATE DEPENDENCY 228, 229, 257
definition 153
- DATE INDEPENDENCE 153, 180, 228
definition 228
- DEDICATED RESOURCES 60, 78, 146, 154, 156, 160, 164, 168, 170, 181, 182, 191, 197, 229, 262, 324
definition 229
- DEFAULT DATA 14, 63, 91, 102, 152, 180, 230, 267, 313
definition 230
- DEPUTY 128, 177, 182, 231
definition 231
- DESIGN FOR REUSE 21, 22, 24, 62, 85, 88, 90, 130, 158, 160, 161, 164, 170, 180, 184, 195, 202, 233, 241, 253, 268
definition 233
- Design Issues
BRITTLE SCRIPTS 147
COMPLEX ENVIRONMENT 150

- DATE DEPENDENCY 153
GIANT SCRIPTS 157
HARD-TO-AUTOMATE 158
HARD-TO-AUTOMATE RESULTS 159
INCONSISTENT DATA 169
INTERDEPENDENT TEST CASES 175
LONG SET-UP 185
MANUAL MIMICRY 186
MULTIPLE PLATFORMS 189
OBSCURE TESTS 194
REPETITIOUS TESTS 195
TOO EARLY AUTOMATION 202
TOOL DEPENDENCY 203
UNAUTOMATABLE TEST CASES 205
- Design Patterns
- ABSTRACTION LEVELS 211
 - AUTOMATE GOOD TESTS 215
 - AUTOMATE THE METRICS 216
 - CAPTURE-REPLAY 220
 - CHAINED TESTS 223
 - COMPARISON DESIGN 225
 - DATA-DRIVEN TESTING 227
 - DATE INDEPENDENCE 228
 - DEFAULT DATA 230
 - DESIGN FOR REUSE 233
 - DOMAIN-DRIVEN TESTING 239
 - DON'T REINVENT THE WHEEL 240
 - FRESH SETUP 245
 - INDEPENDENT TEST CASES 254
 - KEEP IT SIMPLE 256
 - KEYWORD-DRIVEN TESTING 257
 - MAINTAINABLE TESTWARE 268
 - MODEL-BASED TESTING 273
 - ONE CLEAR PURPOSE 276
 - READABLE REPORTS 284
 - RIGHT INTERACTION LEVEL 286
 - SENSITIVE COMPARE 292
 - SHARED SETUP 300
 - SINGLE PAGE SCRIPTS 304
 - SPECIFIC COMPARE 305
 - TEMPLATE TEST 309
 - TEST AUTOMATION FRAMEWORK 312
- TEST SELECTOR 315
TESTWARE ARCHITECTURE 319
THINK OUT-OF-THE-BOX 321
TOOL INDEPENDENCE 322
VERIFY-ACT-VERIFY 326
- Diagnostic 29, 30, 64, 265
First question 8, 73, 132
Fourth question
 - People costs 77
 - Updating the automation scripts too costly 76

Second question
 - Lack of resources 8
 - Lack of support 10
 - Maintenance expectations not met 74
 - Management expectations for automation not met 73

Third question
 - Maintenance costs too high 75
 - Testers don't help the automation team 10

DIE principle 22, 88, 253

DO A PILOT 15, 16, 21, 34, 52, 78, 89, 123, 146, 150, 151, 159, 161, 162, 182, 183, 199, 200, 203, 209, 212, 234, 271, 318

definition 234

DOCUMENT THE TESTWARE 25, 118, 152, 164, 175, 177, 183, 194, 198, 232, 234, 237, 315

definition 237

DOMAIN-DRIVEN TESTING 12, 23, 30, 33, 62, 85, 89, 178, 189, 192, 193, 194, 203, 212, 239, 258, 269, 320, 323

definition 239

DON'T REINVENT THE WHEEL 60, 92, 107, 130, 196, 202, 240

definition 240

DON'T REINVENT THE WHEEL 184, 262

DRY principle 253

DRY Principle 22, 88

EASY TO DEBUG FAILURES 17, 159, 172, 181, 236, 241

definition 241

Execution Issues

FALSE FAIL 153

FALSE PASS 154
FLAKY TESTS 155
INADEQUATE RESOURCES 164
INEFFICIENT EXECUTION 170
INEFFICIENT FAILURE
 ANALYSIS 171
INFLEXIBLE AUTOMATION 174
LITTER BUG 183
MANUAL INTERVENTIONS 185
Execution Patterns
 COMPARE WITH PREVIOUS
 VERSION 225
 EASY TO DEBUG FAILURES
 241
 EXPECT INCIDENTS 242
 EXPECTED FAIL STATUS 243
 FAIL GRACEFULLY 244
 OBJECT MAP 274
 ONE-CLICK RETEST 277
 PARALLELIZE TESTS 279
 PRIORITIZE TESTS 282
 SKIP VOID INPUTS 305
 STEEL THREAD 306
 UNATTENDED TEST
 EXECUTION 324
 VARIABLE DELAYS 325
 VISUALIZE EXECUTION 328
EXPECT INCIDENTS 171, 181, 242
 definition 242
EXPECTED FAIL STATUS 172,
 180, 227, 242, 243
 definition 243
Expert system 104
FAIL GRACEFULLY 40, 156, 180,
 184, 245, 255, 324
 definition 244
FALSE FAIL 227, 230, 247, 255,
 267, 287, 293, 300, 306, 317
 definition 153
FALSE PASS 63, 227, 267, 287,
 293, 306, 317, 327
 definition 154
FLAKY TESTS 168, 230, 245, 255,
 287, 317, 326, 327
 definition 155
FRAMEWORK COMPETITION 130
FRESH SETUP 39, 40, 154, 155,
 157, 160, 170, 173, 176, 180, 245,
 255, 301, 326
 definition 245
FULL TIME JOB 46, 93, 167, 174,
 197, 220, 230, 247, 330
 definition 247
GET ON THE CLOUD 159, 163,
 164, 168, 189, 230, 248, 279, 324
 definition 248
GET TRAINING 68, 85, 131, 177,
 182, 183, 184, 191, 205, 208, 219,
 232, 241, 249, 280
 definition 249
GIANT SCRIPTS 213, 234, 252,
 254, 276, 304
 definition 157
GOOD DEVELOPMENT PROCESS
 17, 21, 60, 62, 85, 86, 149, 152,
 157, 165, 180, 182, 194, 198, 202,
 236, 251, 262, 268
 definition 251
GOOD PROGRAMMING
 PRACTICES 17, 18, 20, 21, 24,
 25, 27, 28, 30, 48, 60, 62, 67, 81,
 84, 85, 86, 87, 149, 152, 157, 161,
 180, 194, 197, 202, 204, 205, 234,
 236, 253, 262, 268, 323
 definition 253
Happy Path 308
HARD-TO-AUTOMATE 76, 225,
 230, 234, 237, 242, 247, 249, 257,
 290, 300, 308, 319, 322, 326, 330
 definition 158
HARD-TO-AUTOMATE RESULTS
 225, 230, 234, 247, 300, 308, 330
 definition 159
Hayku 1, 7, 20, 37, 45, 55, 59, 66,
 71, 83, 95, 102, 114, 122, 129
HIGH ROI EXPECTATIONS 73,
 215, 216, 234, 237, 254, 269, 272,
 281, 295, 300, 308, 313, 323, 330
 definition 160
INADEQUATE COMMUNICATION
 11, 218, 249, 300, 330
 definition 162
INADEQUATE DOCUMENTATION
 75, 234, 238, 298
 definition 163
INADEQUATE RESOURCES 272
 definition 164
INADEQUATE REVISION
 CONTROL 75, 252
 definition 165

- INADEQUATE SUPPORT 10, 77, 79, 271, 272, 281, 292, 300, 303, 311
definition 165
- INADEQUATE TEAM 9, 220, 248, 295, 330
definition 166, 173
- INADEQUATE TECHNICAL RESOURCES 230, 249
definition 167
- INADEQUATE TOOLS 264, 271, 272, 273, 282, 290, 292, 303, 313, 319
definition 168
- INCONSISTENT DATA 75, 230, 234, 247, 300, 330
definition 169
- INDEPENDENT TEST CASES 22, 39, 68, 88, 154, 157, 171, 175, 176, 180, 181, 195, 242, 253, 254, 279, 317, 324
definition 254
- INEFFICIENT EXECUTION 238, 243, 255, 279, 283, 305, 307, 316, 325, 326
definition 170
- INEFFICIENT FAILURE ANALYSIS 74, 225, 227, 242, 244, 247, 257, 277, 284, 308
definition 171
- INFLEXIBLE AUTOMATION 238, 255, 284, 316, 321, 325
definition 174
- INSUFFICIENT METRICS 217, 292, 300
definition 176
- INTERDEPENDENT TEST CASES 38, 189, 223, 247, 255, 277
definition 175
- KEEP IT SIMPLE 21, 22, 27, 63, 88, 90, 151, 152, 153, 158, 159, 172, 188, 195, 198, 242, 253, 256, 267, 285, 322
definition 256
- Keyword
short description 105
- Keyword tests
how not to develop 3
- KEYWORD-DRIVEN TESTING 12, 13, 17, 22, 33, 34, 35, 36, 62, 72, 86, 88, 89, 90, 105, 178, 180, 188, 192, 203, 212, 214, 236, 240, 241, 253, 257, 269, 304, 320, 323
definition 257
- KILL THE ZOMBIES 63, 64, 152, 195, 198, 259, 267, 285
definition 259
- KNOW WHEN TO STOP 115, 182, 200, 206, 207, 218, 260
definition 260
- KNOW-HOW LEAKAGE 91, 103, 104, 232, 238, 250, 279, 300, 313
definition 177
- LATE TEST CASE DESIGN 240, 258, 330
definition 177
- LAZY AUTOMATOR 59, 60, 61, 67, 68, 84, 99, 188, 199, 201, 205, 219, 261
definition 261
- LEARN FROM MISTAKES 17, 152, 169, 193, 195, 198, 200, 201, 207, 208, 236, 263, 291, 299, 302
definition 263
- LIMITED EXPERIENCE 9, 214, 224, 237, 238, 250, 279, 282, 307, 308
definition 178
- LITTER BUG 245
definition 183
- LOCALISED REGIMES 46, 93, 130, 131, 132, 234, 241, 250, 298, 300, 314, 330
definition 184
- LONG SET-UP 223, 301
definition 185
- LOOK AHEAD 127, 147, 264, 314
definition 264
- LOOK FOR TROUBLE 61, 68, 84, 127, 147, 205, 262, 265, 314
definition 265
- MAINTAIN THE TESTWARE 62, 63, 86, 87, 152, 154, 155, 182, 195, 198, 199, 252, 266, 269
definition 266
- MAINTAINABLE TESTWARE 30, 60, 61, 68, 81, 83, 85, 148, 152, 161, 180, 182, 191, 194, 198, 205, 262, 268
definition 268
- Management Issues

- AD-HOC AUTOMATION 145
HIGH ROI EXPECTATIONS 160
INADEQUATE SUPPORT 165
INADEQUATE TEAM 166, 173
INADEQUATE TECHNICAL RESOURCES 167
INADEQUATE TOOLS 168
KNOW-HOW LEAKAGE 177
LIMITED EXPERIENCE 178
LOCALISED REGIMES 184
NO PREVIOUS TEST AUTOMATION 190
OBSCURE MANAGEMENT REPORTS 193
SCHEDULE SLIP 196
SUT REMAKE 200
UNMOTIVATED TEAM 207
UNREALISTIC EXPECTATIONS 208
- Management Patterns
- AUTOMATE EARLY 214
 - AUTOMATION ROLES 219
 - DEDICATED RESOURCES 229
 - DEPUTY 231
 - DO A PILOT 234
 - GET ON THE CLOUD 248
 - GET TRAINING 249
 - KNOW WHEN TO STOP 260
 - MANAGEMENT SUPPORT 270
 - MIX APPROACHES 272
 - PLAN SUPPORT ACTIVITIES 280
 - PREFER FAMILIAR SOLUTIONS 282
 - RIGHT TOOLS 287
 - SELL THE BENEFITS 291
 - SET CLEAR GOALS 293
 - SIDE-BY-SIDE 303
 - TEST AUTOMATION BUSINESS CASE 309
 - TEST AUTOMATION OWNER 313
 - TESTABLE SOFTWARE 317
- Management support
- missing 5
- MANAGEMENT SUPPORT 14, 15, 30, 78, 79, 80, 81, 83, 91, 103, 128, 146, 147, 148, 152, 162, 164, 166, 169, 182, 191, 197, 198, 200, 201, 203, 206, 207, 220, 232, 248, 270, 290, 313, 314, 318, 321
- definition 270
- MANUAL INTERVENTIONS 77, 175, 273, 277, 290, 319, 325, 326
- definition 185
- MANUAL MIMICRY 39, 175, 240, 257, 258, 262, 276, 322
- definition 186
- MIX APPROACHES 16, 169, 186, 235, 272
- definition 272
- MODEL-BASED TESTING 30, 33, 81, 83, 89, 148, 212, 273
- definition 273
- Models
- Dessert model for OBJECT MAP 41
 - Furniture model for keywords 35
 - Hamburger model for ABSTRACTION LEVELS 31
 - Train model for INTERDEPENDENT TEST CASES 37
- MULTIPLE PLATFORMS 213, 313, 322
- definition 189
- Naming conventions 96
- NO INFO ON CHANGES 300, 303, 330
- definition 190
- NO PREVIOUS TEST AUTOMATION 218, 220, 230, 250, 269, 272, 281, 290, 296, 308, 311, 314, 325, 330
- definition 190
- NON-TECHNICAL TESTERS 11
- NON-TECHNICAL-TESTERS 11, 199, 240, 258, 276, 300, 313
- definition 192
- OBJECT MAP 12, 25, 27, 42, 44, 48, 62, 68, 85, 96, 180, 192, 195, 204, 233, 268, 269, 274, 297, 323
- definition 274
- OBSCURE MANAGEMENT REPORTS 73, 264, 284, 300
- definition 193
- OBSCURE TESTS 75, 96, 234, 238, 240, 252, 254, 255, 257, 259, 264,

- 267, 269, 274, 276, 286, 296, 298, 304, 323
definition 194
- ONE CLEAR PURPOSE 40, 157, 180, 188, 194, 242, 255, 276
definition 276
- ONE-CLICK RETEST 172, 186, 242, 277
definition 277
- Organisation of testware 97
- PAGE OBJECT 304
- PAIR UP 87, 177, 181, 183, 206, 232, 250, 252, 278, 317
definition 278
- PARALLELIZE TESTS 171, 181, 279
definition 279
- perception
test automation 3
- PLAN SUPPORT ACTIVITIES 63, 79, 166, 182, 191, 197, 208, 220, 230, 267, 280, 311
definition 280
- PREFER FAMILIAR SOLUTIONS 16, 169, 183, 235, 241, 273, 282, 290
definition 282
- PRIORITISE TESTS 40
- PRIORITIZE TESTS 116, 118, 175, 180, 181, 182, 255, 277, 283, 316
definition 282
- process issues
DATA CREEP 151
- process Issues
INADEQUATE REVISION CONTROL 165
INSUFFICIENT METRICS 176
- Process Issues
AUTOMATION DECAY 146
BUGGY SCRIPTS 149
INADEQUATE COMMUNICATION 162
INADEQUATE DOCUMENTATION 163
LATE TEST CASE DESIGN 177
NO INFO ON CHANGES 190
NON-TECHNICAL-TESTERS 192
SCRIPT CREEP 197
STALLED AUTOMATION 198
TEST DATA LOSS 201
- TOOL-DRIVEN AUTOMATION 204
- UNFOCUSED AUTOMATION 206
- Process Patterns
ASK FOR HELP 213
AUTOMATE WHAT'S NEEDED 217
CELEBRATE SUCCESS 221
CHECK-TO-LEARN 223
DOCUMENT THE TESTWARE 237
FULL TIME JOB 247
GOOD DEVELOPMENT PROCESS 251
GOOD PROGRAMMING PRACTICES 253
KILL THE ZOMBIES 259
LAZY AUTOMATOR 261
LEARN FROM MISTAKES 263
LOOK AHEAD 264
LOOK FOR TROUBLE 265
MAINTAIN THE TESTWARE 266
PAIR UP 278
REFACTOR THE TESTWARE 285
SET STANDARDS 296
SHARE INFORMATION 298
SHORT ITERATIONS 301
TAKE SMALL STEPS 307
TEST THE TESTS 316
WHOLE TEAM APPROACH 329
- READABLE REPORTS 172, 180, 181, 193, 284, 292, 324
definition 284
- REFACTOR THE TESTWARE 63, 147, 152, 181, 182, 194, 196, 198, 256, 267, 285
definition 285
- REPETITIOUS TESTS 228, 241, 286
definition 195
- RIGHT INTERACTION LEVEL 154, 155, 157, 285, 286
definition 286
- RIGHT TOOLS 16, 159, 169, 182, 186, 191, 204, 235, 287, 295, 307, 319
definition 287
- score card 288

- SCHEDULE SLIP 9, 73, 230, 248, 272, 281, 303, 330
 definition 196
- SCRIPT CREEP 96, 238, 252, 254, 257, 259, 264, 267, 269, 272, 274, 286, 296, 298, 317
 definition 197
- SELL THE BENEFITS 15, 79, 80, 166, 169, 176, 182, 200, 271, 291, 308, 318
 definition 291
- SENSITIVE COMPARE 154, 155, 226, 292, 306
 definition 292
- SET CLEAR GOALS 15, 16, 59, 78, 146, 161, 167, 174, 182, 191, 209, 235, 271, 284, 288, 293, 310
 definition 293
- SET STANDARDS 22, 68, 85, 87, 88, 95, 96, 118, 130, 150, 164, 182, 184, 195, 198, 205, 232, 238, 252, 253, 276, 296, 315
 definition 296
 Naming conventions 296
 Organisation of testware 297
- SHARE INFORMATION 12, 46, 80, 93, 95, 115, 128, 131, 150, 151, 154, 159, 160, 161, 163, 165, 166, 170, 176, 177, 182, 185, 190, 193, 201, 203, 206, 208, 209, 218, 230, 232, 248, 271, 280, 284, 288, 291, 294, 298, 302, 314, 318, 322, 329
 definition 298
- SHARED SETUP 41, 185, 223, 247, 300
 definition 300
- SHORT ITERATIONS 17, 80, 86, 96, 166, 182, 190, 196, 208, 236, 251, 292, 301, 308
 definition 301
- SIDE-BY-SIDE 14, 91, 102, 169, 290, 303, 313
 definition 303
- SINGLE PAGE SCRIPTS 25, 158, 180, 195, 233, 242, 304
 definition 304
- SKIP VOID INPUTS 22, 88, 171, 253, 305
 definition 305
- SPECIFIC COMPARE 154, 226, 293, 305
 definition 305
- STALLED AUTOMATION 34, 67, 89, 133, 204, 212, 213, 237, 261, 262, 264, 266, 267, 272, 292, 308, 313, 314, 330
 definition 198
- Standards
 developing 98
- STEEL THREAD 17, 171, 183, 214, 235, 303, 306, 308
 definition 306
- SUT REMAKE 76, 213, 237, 264, 272, 300, 330
 definition 200
- TAKE SMALL STEPS 17, 151, 159, 160, 162, 173, 183, 192, 200, 210, 235, 307, 322
 definition 307
- TEMPLATE TEST 97, 297, 309
 definition 309
- Test Automation
 why do it 49
- TEST AUTOMATION BUSINESS
 CASE 15, 78, 146, 271, 309
 definition 309
- TEST AUTOMATION FRAMEWORK
 12, 13, 23, 65, 66, 69, 70, 72, 90, 91, 102, 117, 118, 162, 169, 189, 193, 200, 216, 219, 231, 239, 258, 271, 277, 283, 290, 303, 305, 312, 315, 321, 323, 328
 definition 312
 Kitchen model 92
 reasons not to develop in house 92
 why develop in house 104
- TEST AUTOMATION OWNER 67, 78, 84, 127, 130, 145, 147, 184, 191, 199, 205, 220, 271, 313
 definition 313
- Test Automation Patterns
 difference from unit test patterns 7
 number of 7
 short description 7, 103
- TEST DATA LOSS 75, 234, 241, 252, 254, 262
 definition 201
- Test Definition 97

- TEST SELECTOR 97, 117, 118, 171, 174, 180, 182, 238, 283, 297, 315
definition 315
- TEST THE TESTS 87, 149, 154, 155, 157, 252, 316
definition 316
- TESTABLE SOFTWARE 17, 159, 169, 186, 214, 236, 317
definition 317
- Testing knowhow
Equivalence classes 56
- TESTWARE ARCHITECTURE 14, 17, 21, 29, 30, 34, 67, 81, 83, 84, 89, 91, 102, 148, 150, 175, 180, 182, 203, 205, 212, 219, 220, 236, 310, 312, 319
definition 319
- THINK OUT-OF-THE-BOX 60, 151, 159, 160, 188, 189, 206, 256, 262, 321, 325
definition 321
- TOO EARLY AUTOMATION 11, 213, 214, 216, 237, 240, 259, 272, 300, 321, 330
definition 202
- TOOL DEPENDENCY 13, 75, 76, 213, 240, 276, 290, 323
definition 203
- TOOL INDEPENDENCE 25, 26, 34, 43, 61, 67, 84, 89, 162, 180, 195, 204, 205, 212, 233, 262, 275, 322
definition 322
- TOOL-DRIVEN AUTOMATION 66, 67, 69, 72, 84, 221, 254, 262, 266, 269, 298, 314, 321, 330
definition 204
- UNATTENDED TEST EXECUTION 171, 175, 181, 182, 186, 192, 322, 324
definition 324
- UNAUTOMATABLE TEST CASES 261, 272, 279, 281, 300, 322, 330
definition 205
- UNFOCUSED AUTOMATION 218, 261, 264, 272, 330
definition 206
- UNMOTIVATED TEAM 11, 222, 248, 251, 264, 281, 300, 303, 330
definition 207
- UNREALISTIC EXPECTATIONS 10, 73, 77, 123, 133, 237, 271, 292, 296, 299, 300, 308
definition 208
- VARIABLE DELAYS 157, 159, 171, 180, 181, 186, 243, 325
definition 325
- VERIFY-ACT-VERIFY 155, 157, 326
definition 326
- VISUALIZE EXECUTION 97, 298, 324, 328
definition 328
- WHOLE TEAM APPROACH 46, 58, 67, 69, 78, 84, 93, 95, 96, 146, 160, 161, 162, 163, 167, 170, 174, 178, 182, 190, 191, 197, 199, 200, 203, 205, 206, 207, 208, 329
definition 329