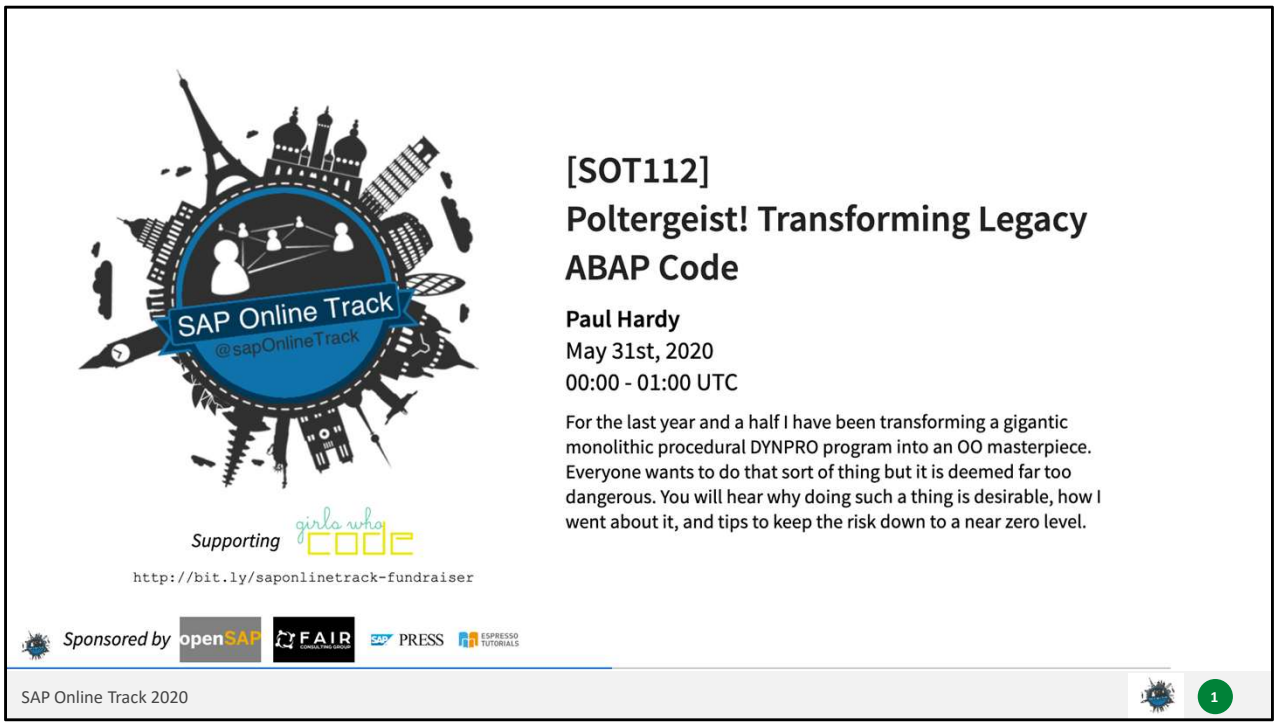


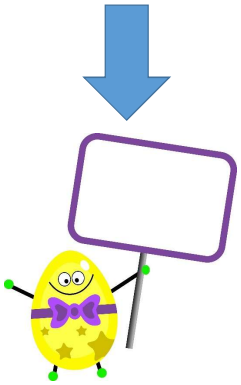
Technical note – not for the audience in the room on the day of the talk but for anyone reading these slides afterwards (even if they are one and the same person). Here is a blog I wrote about how to do presentations:-

https://blogs.sap.com/2020/03/18/present-arms/

A lot of the notes section for each slide will not get "read out" to the audience (no-one should ever read out their notes) they are intended as a guide to the speaker (me) and speakers in general.

For example - this first slide is the one that is on the screen before the speaker starts speaking. In big events it is on the screen as the audience arrives (if anyone turns up, that is)

Watch out for the EGG MAN!



https://blogs.sap.com/2020/05/25/rockin-all-over-the-sap-world/

*NB: After you open your mouth for the first time in a presentation you have **thirty seconds** to hook the audience. If you don't get them by then they will spend the rest of the presentation daydreaming or playing with their mobile phone, or worse. Therefore the opening is the most important thing you have to think about because if you don't get that right then nothing else matters at all.*

It does not really matter HOW you hook the audience just that you do. Just do something unusual or unexpected or just plain different to what everyone else does. And you have to keep changing it.

For this one I am going to surprise the audience with the Egg Man.
Throughout this presentation the egg man shown here will reveal three secret words on his placard.

Make a note of all three words and then try to be the first to post the three words phrase on the SAP Community Network blog about this presentation.

Then I will say I am going to summarise the purpose of the presentation in the form of a Haiku and do just that.

Technical Note: This is the “tell them what you are going to tell them” slide – the message of the entire presentation. It has to be one sentence only. A human brain cannot process more than three points at once, and furthermore those three points all have to be related to one single message. Ideally that message should be ten words or less.

In this case the message is:-

“Old Monolithic ABAP Code can be Safely Changed”. That is eight words, well within the limit!

To hook the audience I won’t say that directly but tell them I am going to summarise the presentation in the form of a Japanese Haiku, and then do just that.

Definition of Haiku is : a three line poem where the 1st and 3rd lines are 5 syllables and the 2nd line is 7 syllables. Usually mentions an aspect of nature and/or a season at some point. The first line is the setting, the second two describe the subject and the action.

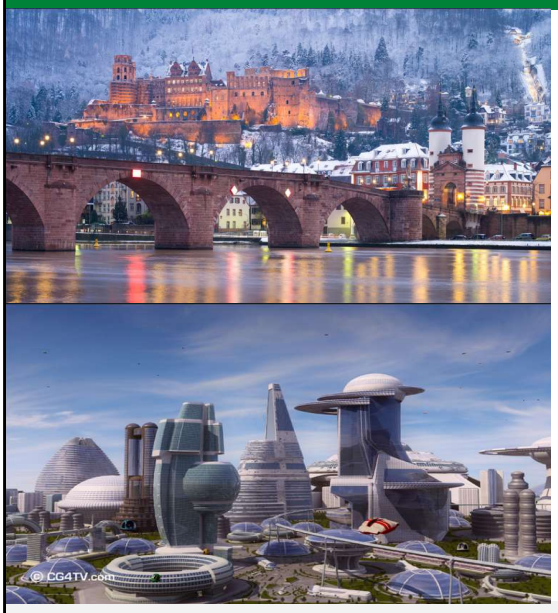
Brittle with frostbite -
Your fragile legacy code
Can be safely changed

Technical Note: It has been said that the vast majority of presentations are intended to make the audience want to do something as a result and so should follow this pattern:-

1. *The present situation and why it is bleak*
2. *Paint a bright future of a possible future*
3. *Tell the audience how to get to that bright future*

This will be an incredibly technical presentation, so the vast bulk will be in the last third. The first two will be my first point, the last one will be my second and third points (theory and practical instructions on how to get to that bright future).

What You'll Hear



- About Heidelberg Cement
- The Problem
- How to fix The Problem – in theory
- How to fix The Problem – in practice

SAP Online Track 2020

4

Today you will hear about…..

- What in the world legacy code is, why most of your code is legacy code, and why that is a bad thing
- The academic theory of how to turn legacy code into good code without destroying your business
- The practical reality of how to turn legacy code into good code without destroying your business

Transition – I am doing this for real, in real life. I did it in Germany for a year and a half, I am now doing it in Australia. If I stuff it up, my company goes down the gurgler. That company is….

Technical note follows – not for the audience, but for anyone reading the notes on these slides.

Notice the word YOU in the heading. The focus of any presentation has to be on the people in the audience. They have to get something out of it and so the focus cannot be on how wonderful the speaker or their company or whatever it is they are talking about is so wonderful, it has to be as if you were talking to one single person with the sole aim of

*making them better off in one way or another e.g. my company has done this great thing, look how much it helped us, you can go back next week and suggest doing this great thing at **your** organisation and **you** personally will get (a huge pay rise and/or promoted and/or knighted).*

The logic above, and the structure of my presentations comes from Toastmasters International. Every speaker at every conference should take note of the advice they have to offer.

One Point the Slide is trying to make:- Tell audience members how they personally can relate to the problems faced by Heidelberg Cement

HeidelbergCement is one of the world's largest building materials companies - which means it is one of the market leaders in producing cement, aggregates (rocks from quarries) and ready mixed concrete. It has around 58,000 employees who work at more than 3,000 production sites in around 60 countries on five continents.

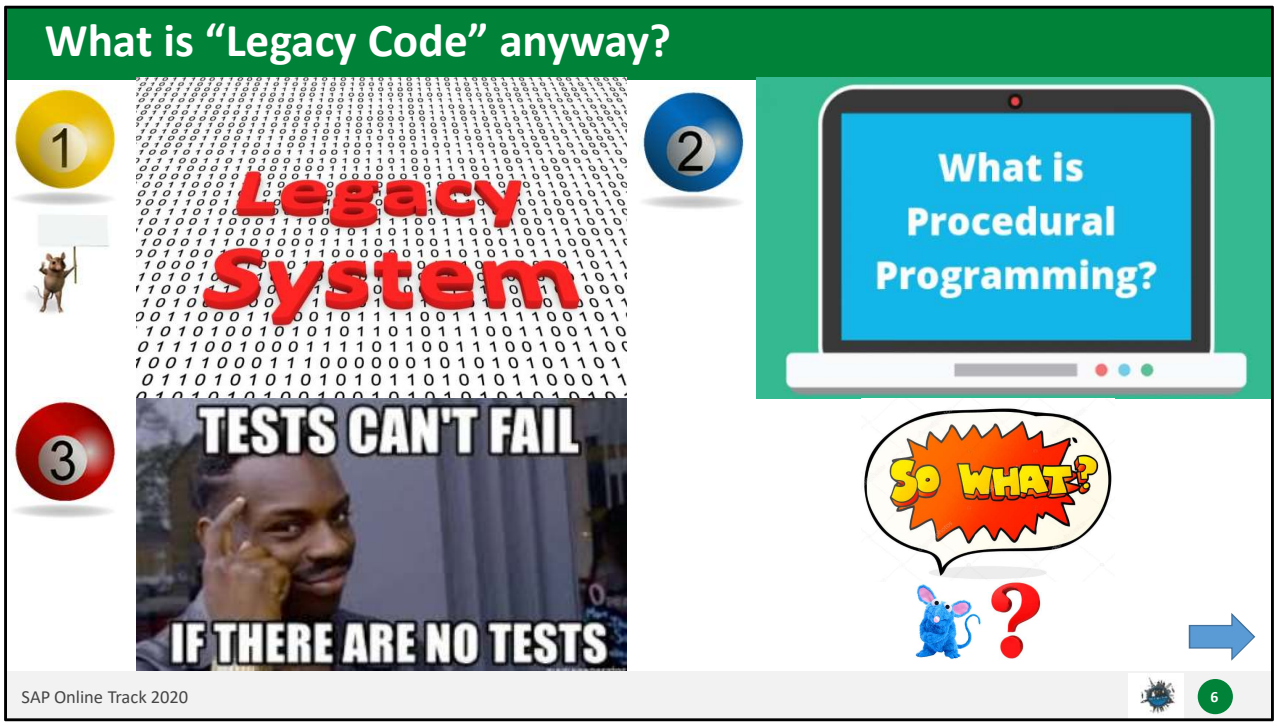
No doubt you could not care less about any of that, but the important things from your point of view is that:-

1. Heidelberg Cement runs SAP: just like your organisation.

2. We depend on an enormous amount of Z code to give us a competitive advantage: just like your organisation. We have to keep pumping out new Z and improved Z code as fast we can to keep our lead, and bring new countries onto SAP.

3. In the near future we have to move all that Z code onto S/4 HANA: : just like your organisation.

Transition - This brings us to ***THE PROBLEM*** of legacy code which I personally at my company, am working on.



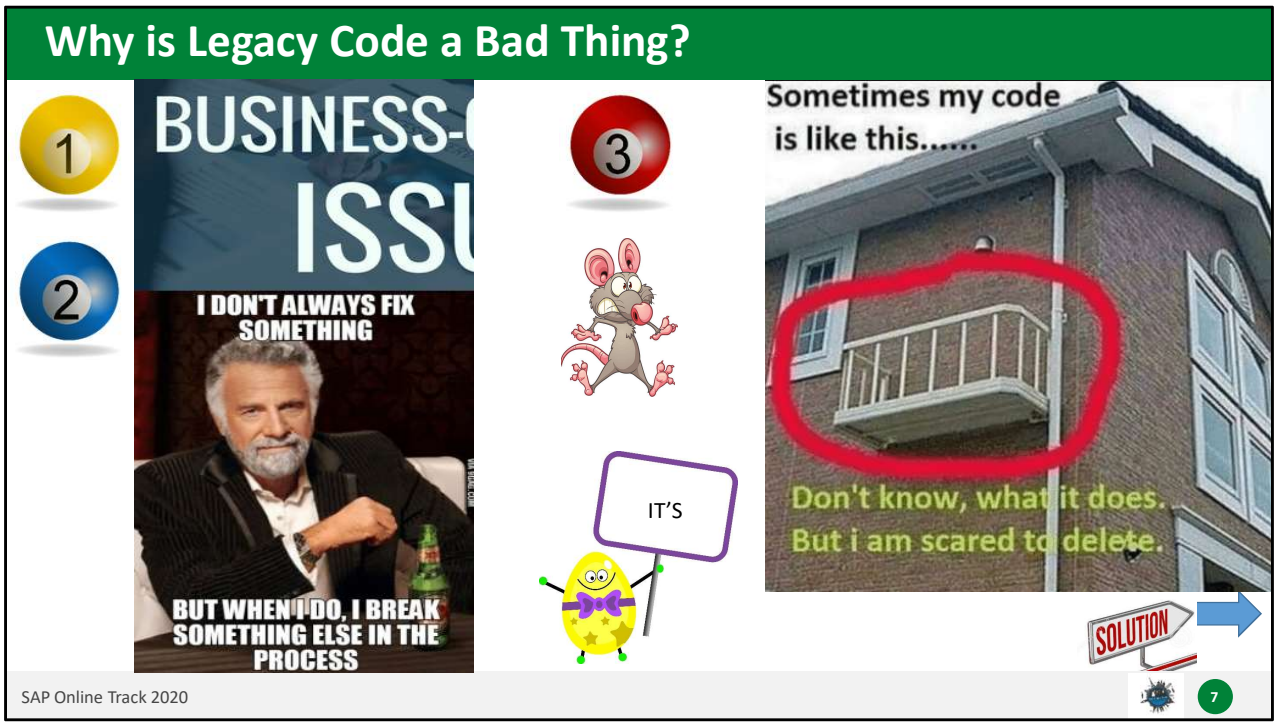
One Point the Slide is trying to make:- To dispel incorrect notions of what people think "Legacy Code" means.

Legacy Code is NOT:-

1. Code from the system you had before SAP i.e. "The Legacy System". You do not use that code any more, so that cannot possibly be what we are talking about here.
2. Procedural Code. It turns out any sort of code can be "Legacy Code" – you could have OO code which is Legacy Code and Procedural Code which is not.
3. The actual definition of "Legacy Code" is very simple indeed – it is code without automated unit tests.

P.S. – A mouse with a placard means that when you get the PDF of this presentation after the event there will be an extra slide with a link to more reference material regarding the subject at hand. The mice without placards have just come along to keep the others company.

Transition:- Now you know what Legacy Code is, and that most of your code is like that, why is that a bad thing?



One Point the Slide is trying to make:- Without tests, you cannot tell if your new change breaks something existing.

1. The sort of huge monolithic applications written in Legacy Code tend to be Business Critical.
2. You cannot make a change because it will break something else and you do not know what.
3. You cannot clean up old code because you are too scared it will break something.

Transition:- It does not have to be like this. This problem is so widespread there is a proven methodology for fixing it.

What You'll Hear

- About Heidelberg Cement
- The Problem
- How to fix The Problem – in theory
- How to fix The Problem – in practice

Transition – you will hear two sides to this coin. The theory that you will find in a famous book on this subject, and how I have actually been doing this in the real world in my ABAP system at work.

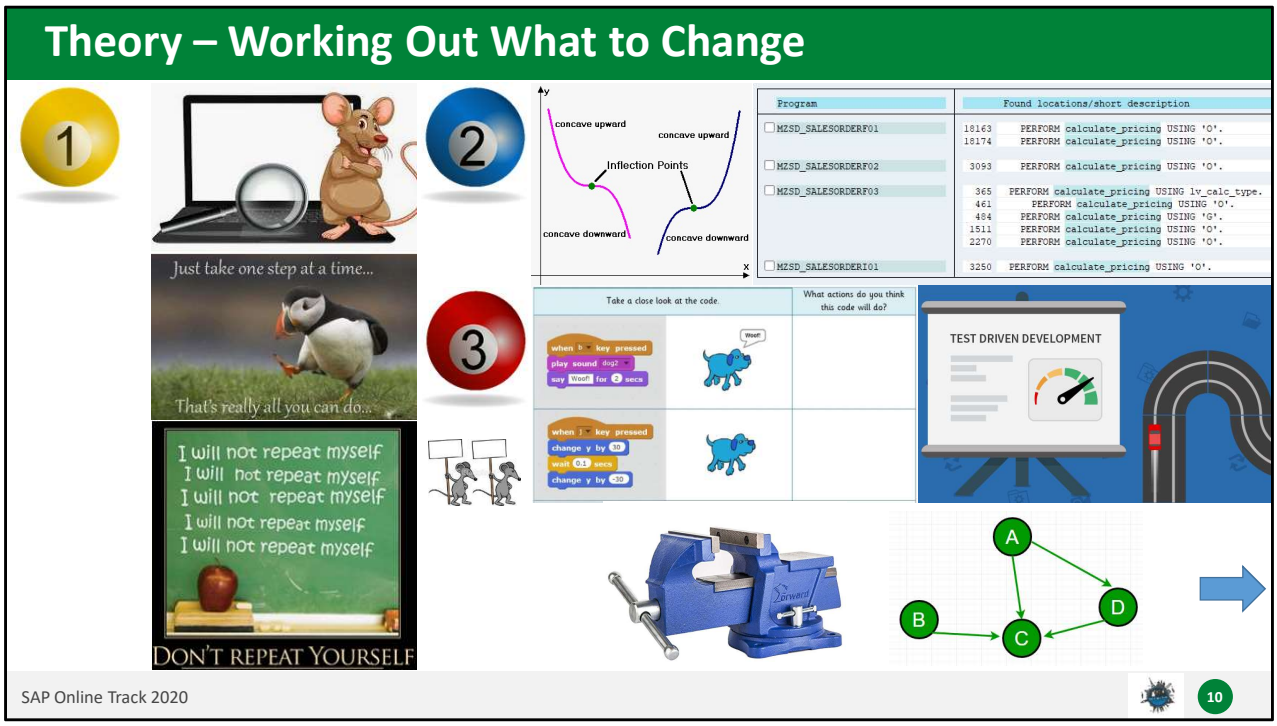
One Point the Slide is trying to make:- Say why Michael Feathers book has been ignored by ABAP Programmers.

1. Many years ago USA author Michael Feathers wrote the book “Working Effectively with Legacy Code” which was all about the best way to add tests to existing code (i.e. stopping it being Legacy Code) with the least risk possible.
2. Since the examples were in Java or similar languages (i.e. not ABAP) and most of the examples started off with Object Orientated code, and 99.9% of the “Legacy Code” in most people’s systems is procedural, this led to ABAP programmers not being interested *at all* in this concept.
3. In 2017 a big bank in Australia wanted to get on top of its Legacy Code and so wanted to hire Michael Feathers. They had three ERP systems, one of which was SAP. Michael Feathers had never even heard of ABAP so I was approached – as I was the only ABAPer in the world squawking about test driven development on the internet - by a consultant to work with MF and convert his examples into ABAP to show to the bank. I talked to Mr. Feathers on the phone, exchanged emails, everything was going great but then the bank got cold feet and decided everything was fine the way it was. I was gutted, that would have been really good.

Transition:- In Michael Feathers book there were a series of steps you had to follow when

working with Legacy Code. In the next few slides you will see how I translated those into ABAP for the bank people who never got to see them….

*Technical Note:- Point Three is a **STORY** something that actually happened to me in real life, and the end result has relevance to the audience (otherwise don't bother telling it). Stories are the "secret sauce" to presentations but they have to be real i.e. this actually happened I did not make it up or even worse take someone else story and pretend it happened to me (horrible though common mistake some presenters make).*



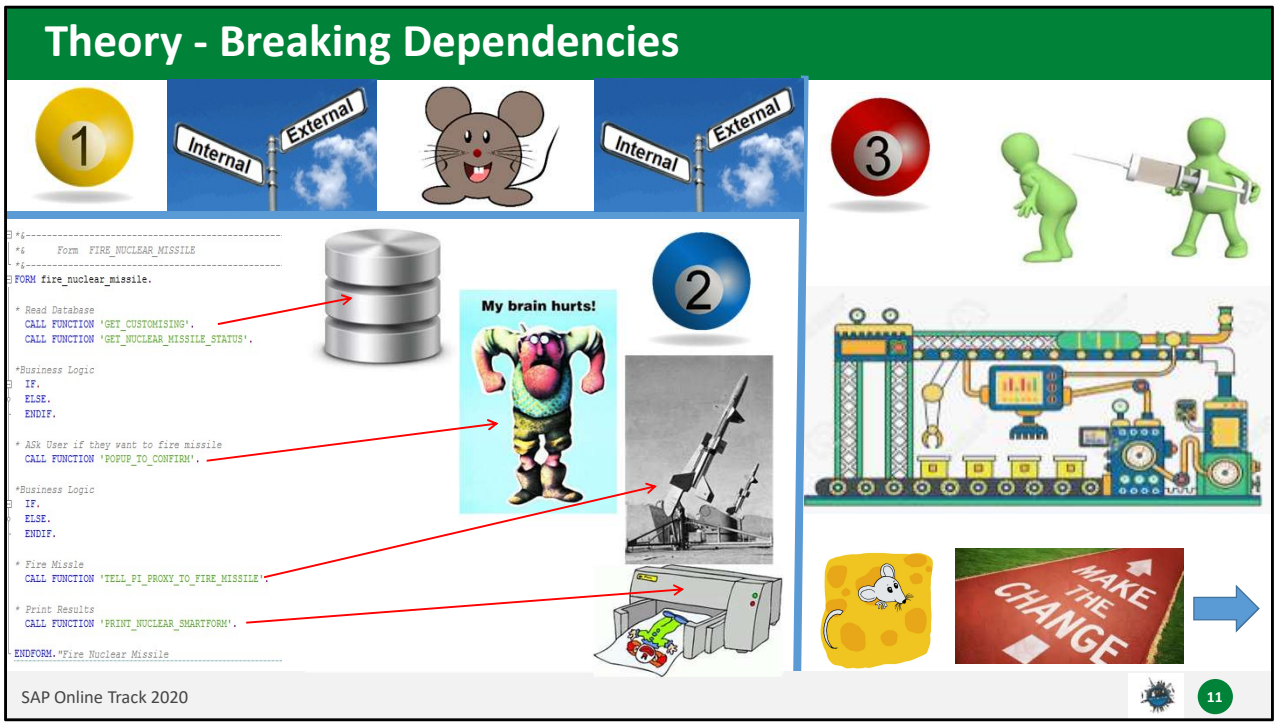
One Point the Slide is trying to make:- How to prepare for the changes you need to make.

Working out what exactly you are going to change

1. Identify Change Points i.e. what code do you change. Break your scope change request into small chunks so you only change one “behaviour” at a time. If you have to change ten different areas of the program to change one behaviour (e.g. new IF or CASE branch in then different places) then that is a clear sign of duplicate code that needs to be encapsulated.
2. Find an Inflection Point. Conceptually an “inflection point” is like the flat bit on a roller coaster ridge before you drop down a hundred feet in 5 seconds. In ABAP this equates to the highest level in the call stack which will be affected by the change and then move it on to the rest of the program. You need to find this because that routine (be it FORM routine or method) is what you are going to write a unit test for. Use the “where-used” list for the routine you are thinking about changing and work upwards. You find an inflection point when (a) the routine at hand is used by two or more routines or (b) the routine at hand is used by something outside the program e.g. function module or public method that could be called by anything.
3. Cover the Inflection Point. This means writing a unit test for the new behaviour ***before*** you make the code change. The specification will say the program currently does *THIS*

and it should be doing *THAT*. Your test will initially verify that the program is in fact currently doing *THIS*. Normally you verify the current behaviour via manual testing. This is important because as you will see you will most likely need to change the code before you even think about changing the behavior and you will want to ensure your changes do not break anything. Michael Feathers calls this putting the existing code in a “vise” (or “vice” if you are English).

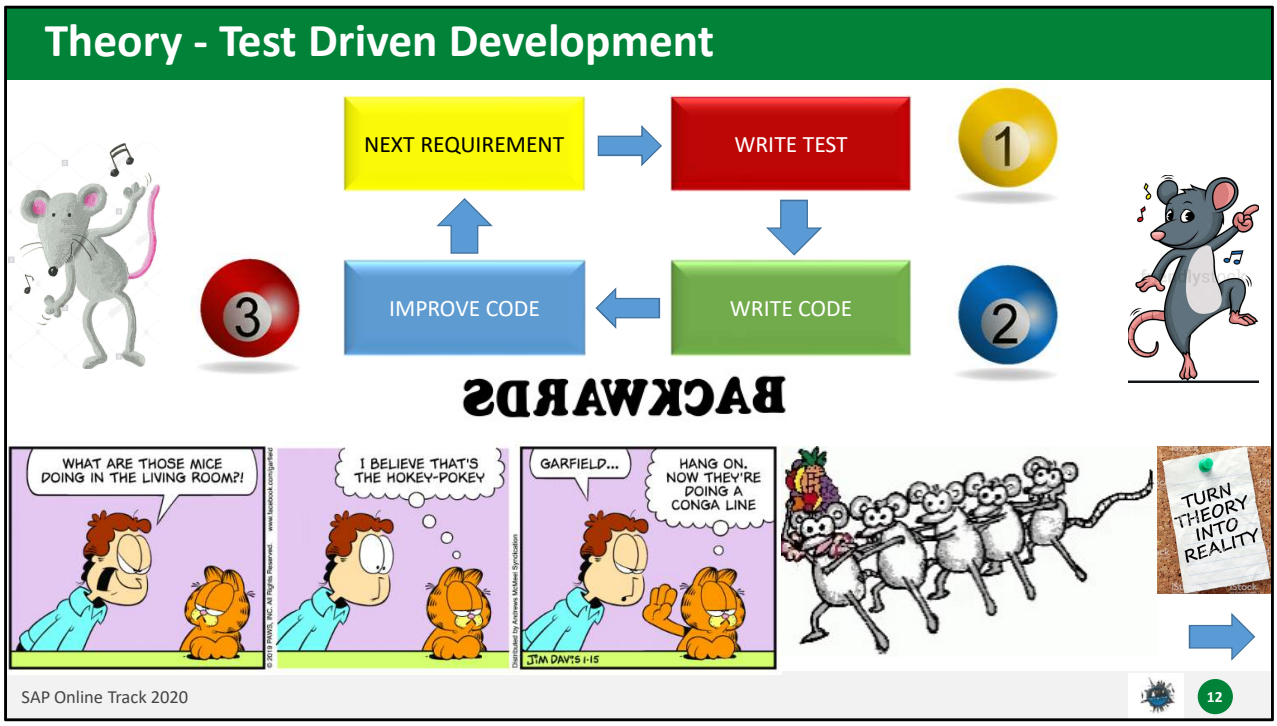
Transition:- you need to make changes because you cannot write a unit test for the code you want to change because it is crawling with *dependencies* which need to be broken.



One Point the Slide is trying to make:- What are dependencies and how to break them.

1. In a traditional OO class there are two sorts of dependencies – objects passed in via parameters (external dependencies) and objects created directly within the class. In both cases these need to be replaced by always getting objects via a factory, which you will hear about in a minute. This is not usually relevant to ABAP where the vast bulk of programs are procedural.
2. In a traditional procedural program all the dependencies are direct calls to the database, UI layer, printer, external system and so on. You cannot write a unit test for a piece of code that has “unbroken” dependencies. Such code can only be manually tested as it needs a real database, user etc. These need to be replaced by equivalent calls to specialised classes which then have their instances created by a factory.
3. A factory class returns a class that uses the real dependency (database or whatever) but during a unit test returns attest double that has been “injected” by the unit test framework in the SETUP method.

Transition:- Once you have made the code testable and ensured you have not broken anything in the process then it is time for the *real* work to begin.



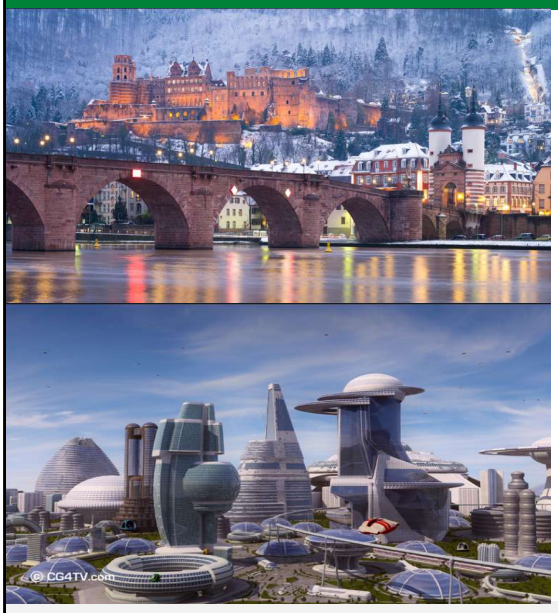
One Point the Slide is trying to make:- To explain the process of Test Driven Development

This is only mentioned briefly in Michael Feathers book, yet it the cornerstone of the whole process. Traditionally you do tests after you have written the code. Now you must change the way you have always programmed and do everything backwards.

1. With TDD you write the test *first* – this is the so called "red phase". You need to have a failing test before changing anything otherwise you will have no idea when you have finished!
2. Then you write (or change) just enough production code to make the test pass – the "green phase".
3. Then you improve the quality of the code you just wrote – the "blue phase". This whole approach forces you to write SOLID programs.

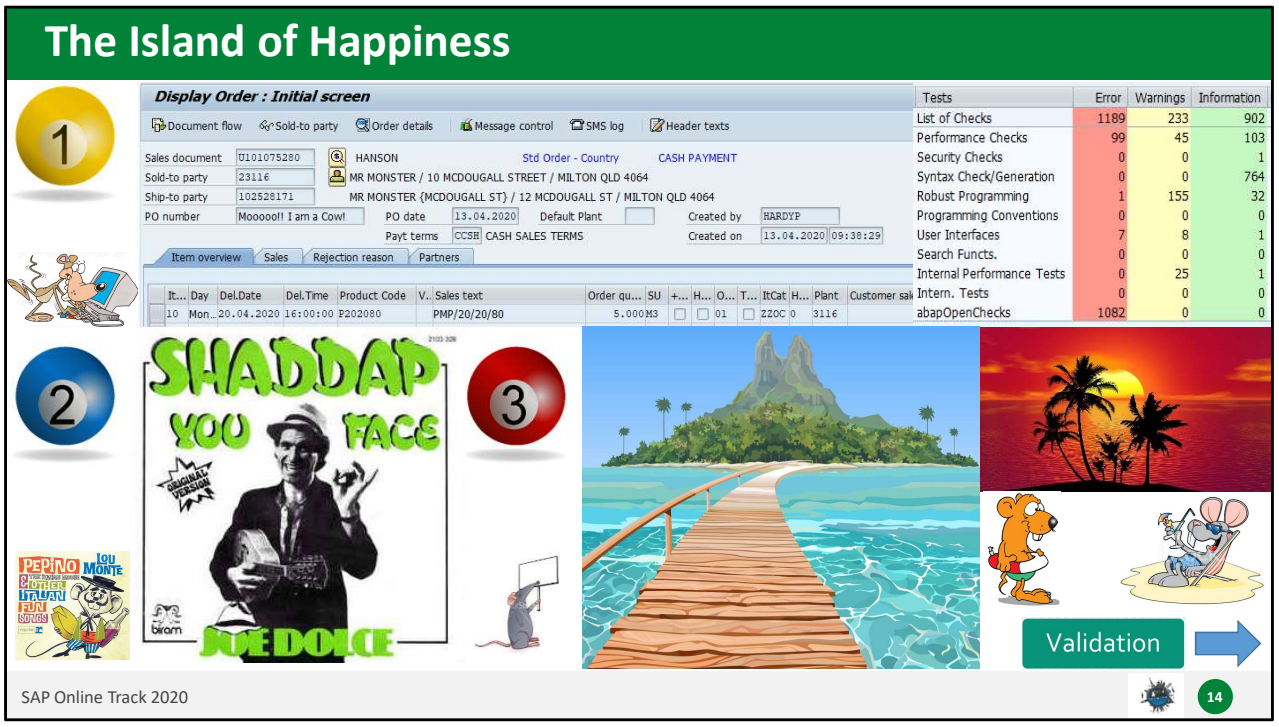
Transition:- That is the theory of what needs to be done, loosely based on the book by Michael Feathers. Now is the time to turn theory into to reality…

What You'll Hear



- About Heidelberg Cement
- The Problem
- How to fix The Problem – in theory
- How to fix The Problem – in practice

Transition – I read the book about Legacy Code many years ago, but only recetntly am I really doing this in practice…

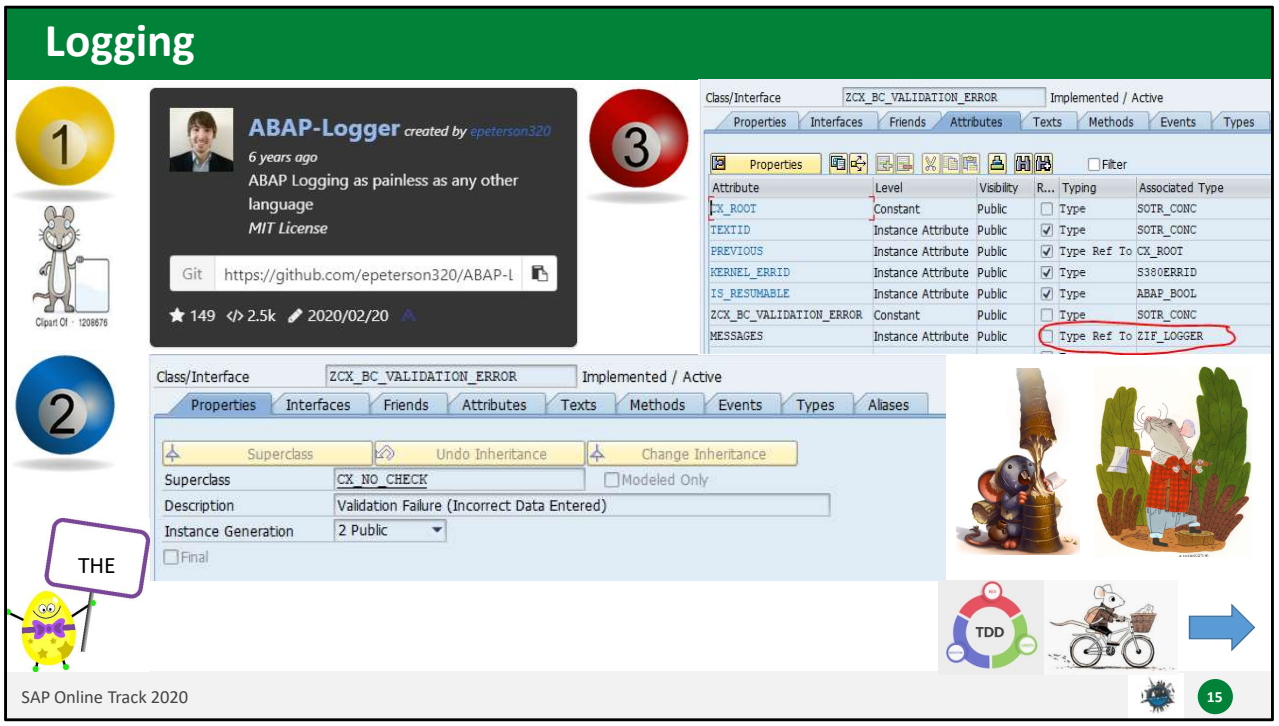


One Point the Slide is trying to make:- Initial steps to create an “Island of Happiness” to be used by the sample application.

1. Introduce guinea pig application – this is a custom front end for the VA01 sales order transaction. The twin purpose is to capture a heap of Z fields, and to enable order takers to create sales orders like lightning. This application is 20 years old (with more code added every week of that time) 100% procedural DYNPRO and a fragile, rotting, business critical mess. I have to change it every month, so it gets worse every month – now is the time to turn that around.
2. Create Interface for Model Class. To start off with all new business logic (derivations / validations) will be in a public method exposed by an interface. This was the least intrusive way I could think of in regard to adding new behavior. All classes should have their public methods defined via an interface - it is good OO design.
3. Create Model Class. I then created the model class which is going to be the “Island of Happiness”. All code written there will be done in a TDD fashion, be guaranteed to work via automated regression tests and have no code inspector errors or warnings. The bridge in the picture represents the fact the current application is going to access the island yet be 100% separate from it.

Transition:- The very first new behaviour I had to add was a validation check to prevent

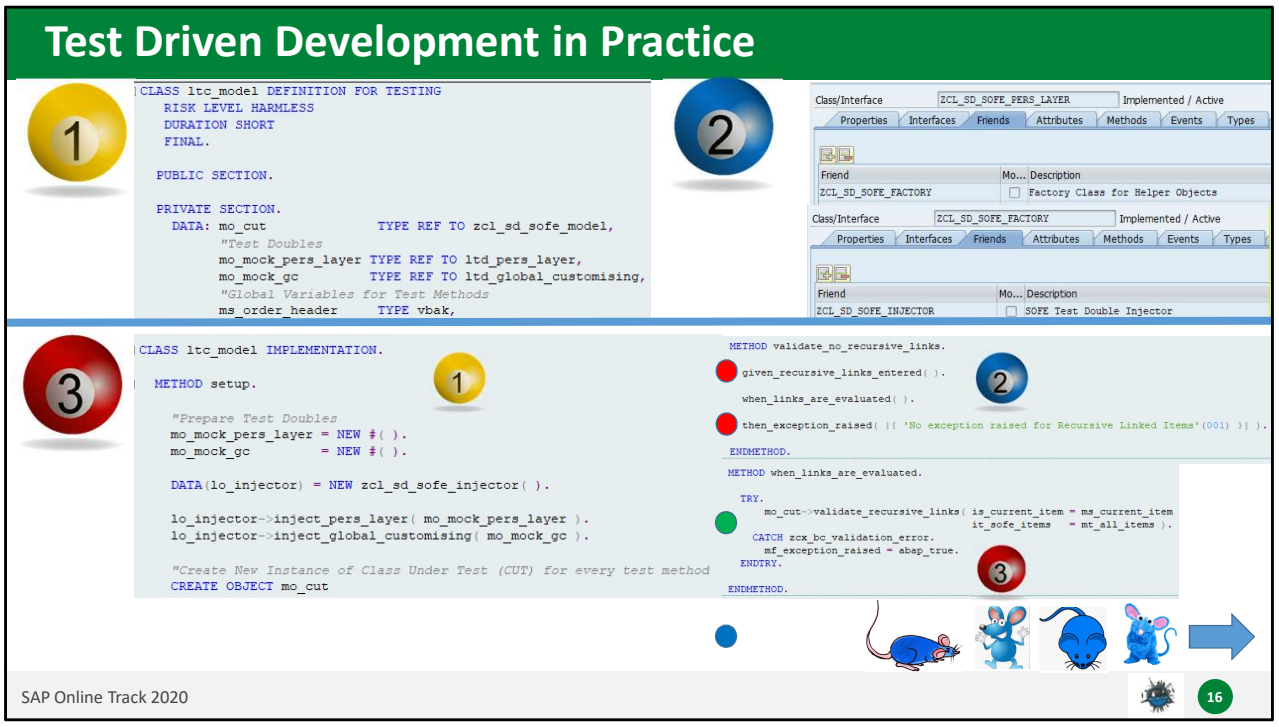
some crazy input certain order takers had been entering. So, I had to work out the best way to get the entered data to the island and send back an error.



One Point the Slide is trying to make:- How to send error messages from the “Island” to the monolithic application.

1. Download ABAP Logger. A model class has no UI and so cannot issue messages itself. You want to send out some sort of error object and let the calling application decide what to do with the errors. As I am no fan of re-inventing the wheel, I thought why not use the open source ABAP Logger on GitHub?
2. Create validation exception. I always like to create my own exception classes as opposed to re-using standard SAP ones. This one is designed for use with all DYNPRO programs which have CRUD methods for “objects” e.g. sales orders, customers, pieces of equipment. Thus, I gave it a generic name. I am also a big fan of “No Check” exceptions where you just hurl the exception into the wild blue yonder and hope it gets caught somewhere up the call stack.
3. Link logger to exception. The model class will transfer the error information into an instance of the logger class (it accepts virtually any error format – messages, BDC errors, BAPI return structures etc.) and then that instance is passed to the calling application as an attribute of the exception.

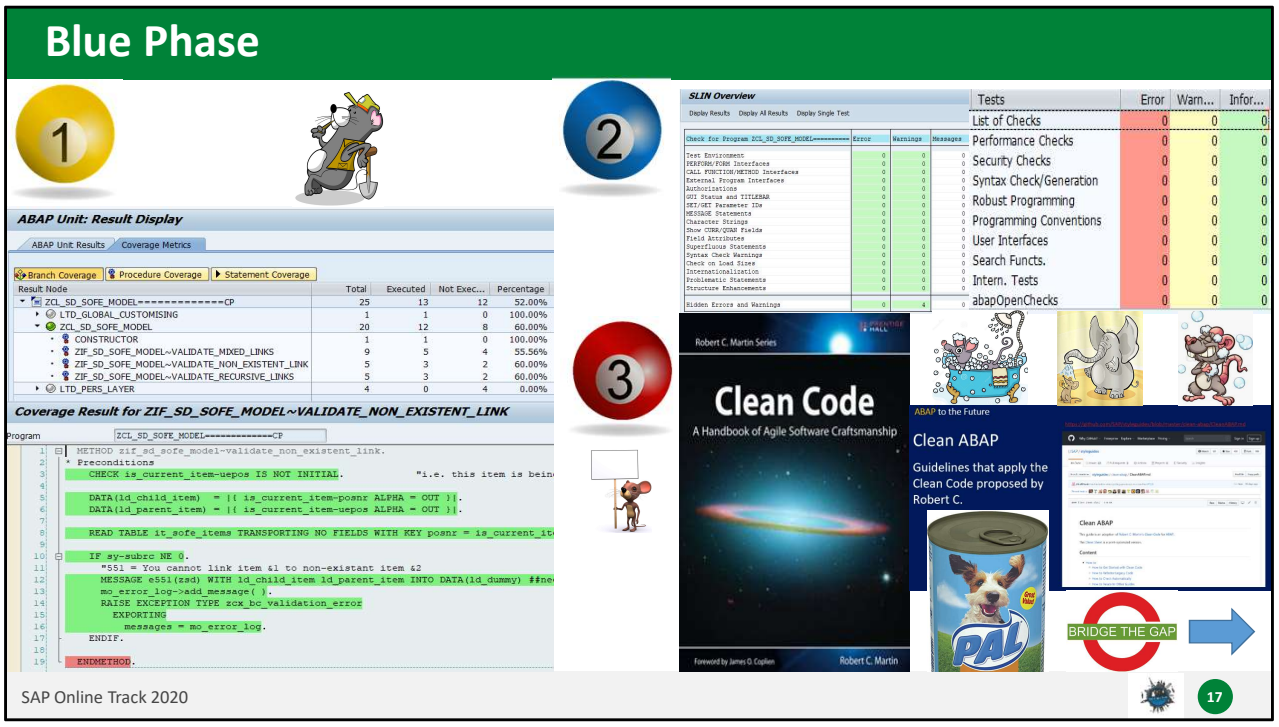
Transition:- You now have enough jigsaw pieces in place to start the TDD cycle which will result in your making your code change…



One Point the Slide is trying to make:- To show screen shots of the actual Test-Driven Development process

1. Before writing a single line of production code in the model class you need to create a test section first for test doubles and the test class itself.
2. You need a factory & injector class for next unit test. The dependency classes e.g. persistency layer are friends with the factory and can only be created by the factory. The factory is friends with the injector so it can have test doubles injected during unit tests.
3. The injection of test doubles happens in the SETUP method run before every test. The unit test method for the new behaviour is written using the GIVEN / WHEN / THEN structure. Make sure it fails by adding the THEN assertion first. At that point you have completed the RED phase. Next in the GIVEN method pass in the input data. Finally, in the WHEN method write a call to the production method (which might not even exist yet) and write that method, adding code until the test passes.

Transition:- Once the test passes you have gone through the RED (failing test) and GREEN (passing test) phases but it is vital you do not forget the last phase…



One Point the Slide is trying to make:- Do not Ignore the “Blue Phase” – it is just as vital as the other two phases.

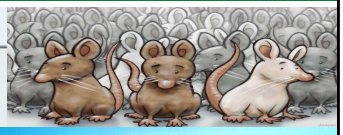
Once the test passes it is time for the “Blue Phase”

1. Run the unit tests with coverage to make sure you have covered every possible scenario i.e. every branch in the conditional logic. Sadly, the tool is a bit buggy, so will give you (say) 66% coverage when in fact you have 100% coverage for a method.
2. Run the extended program check and code inspector, and either fix everything or flag the false positives as such.
3. SAP provide a set of “Clean Code” automated ATC checks you can download from GitHub (Code Pal). Work through those then apply any “Clean ABAP” Principles for which automated tests do not work.

Transition:- The result is that you are 100% sure all the code in the “Island” not only works but is of the highest possible quality. After all, shoddy fragile code can work fine but it is not very future proof. When the Blue Phase is over you can add the new behaviour to your main application.

Link the Island to the Continent


```
MZSD_SALESORDERTOP                    Active
* MVC Objects
DATA: go_sofe_model          TYPE REF TO zif_sd_sofe_model,
      go_st_back_screen_view TYPE REF TO zcl_sd_sofe_view_st_backs.
```



Screen number 101 Active

Attributes | Element list | Flow logic

```
1  *---- P B O ----*
2  PROCESS BEFORE OUTPUT.
3    MODULE create_objects.
```


```
FIELD t_vbap-uepos MODULE validate_links ON REQUEST.

FORM validate_links.

  TRY.
      go_sofe_model->validate_recursive_links(   is_current_item = t_vbap
                                                 it_sofe_items   = t_vbap[] ).
      go_sofe_model->validate_non_existent_link( is_current_item = t_vbap
                                                 it_sofe_items   = t_vbap[] ).
      go_sofe_model->validate_mixed_links(       is_order_header = vbak
                                                 is_current_item = t_vbap
                                                 it_sofe_items   = t_vbap[] ).
    CATCH zcx_bc_validation_error INTO DATA(lo_error).
      lo_error->messages->output_messages( ).
  ENDTRY.

ENDFORM.
```

One Point the Slide is trying to make:- How the main (legacy) application calls the island of Happiness" in a safe way.

1. Add new global object to DYNPRO program. It is just one more global variable alongside the ten thousand that already exist. It will not be lonely.
2. Add "create object" module to both initial screens. As soon as the very first screen pops up to a user the model class is instantiated.
3. Embed call to model class in PBO or PAI modules. In this context the DYNPRO acts as the controller (and view). For a validation a PBO module calls a FORM routine which invokes the logic in the model class. The exception object is passed back containing the log, which in this case is output using MESSAGE statements just like the other errors in the existing DYNPRO validations.

Transition:- If you repeat this process for every new behaviour or extract the logic to the model when changing existing behaviour, then eventually all the (used) logic will be in the model and the legacy application will wither on the vine. However, that is a very long process and right at the start you have a big problem….

One Point the Slide is trying to make:- Say how what you have done looks ridiculous and your colleagues will all laugh at you.

1. Though what you have done with the “island of Happiness” is the correct thing to do and if done properly will transform your legacy code into wonderful modern code there is a problem.
2. Especially at the start it takes a load of extra classes to be created and linked together, a whole new framework, not to mention unit tests, and all for what could be changing one or two lines of code. That is going to look *RIDICULOUS* and a complete waste of money i.e. it takes days instead of minutes.
3. Chances are unless you explain very carefully to your colleagues what exactly it is you are doing and why they will take one look at what you have done and conclude you are mad and/or stupid and laugh their heads off at you.

Transition:- Today we will end on what exactly you need to do to avoid that … but first …

Questions …. Ask Them Online via Discord

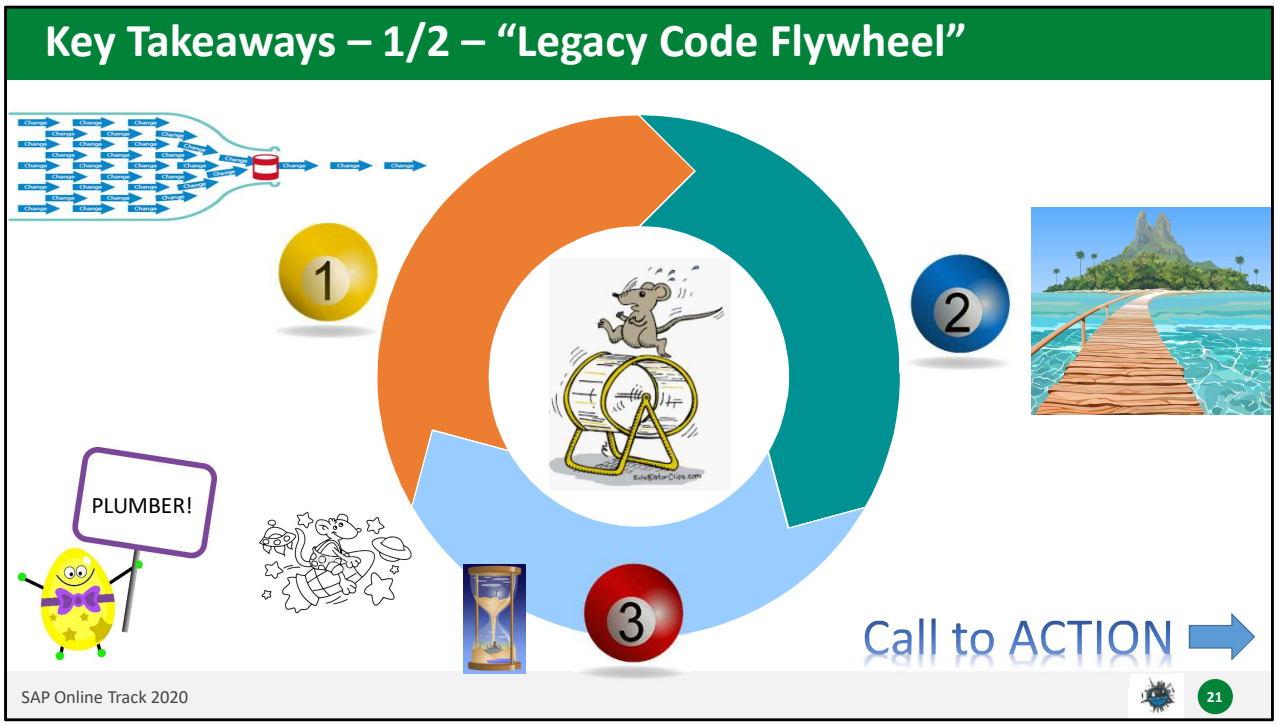
Technical Note : Toastmasters say do not put questions right at the end, but just before. This means the presenter has the last word. It has been said people only remember the first thirty seconds and the last thirty seconds of a presentation, so you do not want the last 30 seconds to be someone asking "does ABAP have IF statements?" or some such (an actual question BTW).

Some people say as soon as the Q&A start they begin packing up and look for a way to make a sneaky exit, so tell them there is something else coming up.

In fact I am not going to take any questions at all now. I have been in the audience often enough to know it is just a painful exercise at any given moment for the person.

Instead I will now talk about HOW to ask questions and THEN I will do a final summary. For those interested in more detail we can keep the discussion going in one of the unconference channels, after the session.

Transition : - let us move straight on to the Wrap-Up

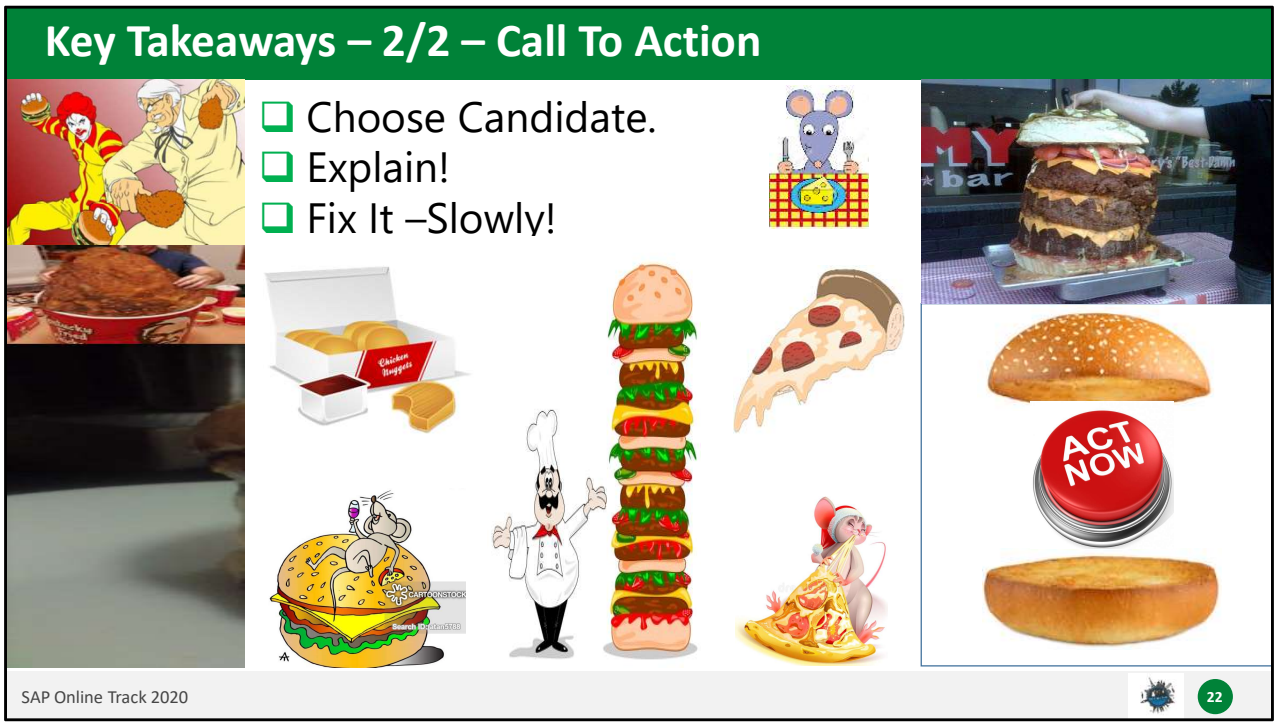


One Point the Slide is trying to make:- To explain the "Legacy Code Flywheel"

Given that anything in the "island" can be changed quickly and easily and the "Legacy Code" cannot then….

1. The more changes you make (properly) in any one cycle….
2. Then higher the %age of code is in the "Island of Happiness" and so..
3. The more changes you can make in the next cycle in the same space of time and so…. (return to step one)

Transition: - Call to Action



Technical Note : This is the final slide shown. This should be a call to action. Therefore the takeaways should not be what you (person in the audience) learnt but what you can do RIGHT NOW to improve life for yourself or your organization in some way. Once again – the call to action should be ten words or less. Any more and your message does not have the required clarity.

DON'T READ OUT WHAT'S ON THE SLIDE. Just mention that "you" can start putting what has been presented today into action straight away, as soon as you get back to your desk.

Here I have just six words – the audience can read them in a second or two, while I am drawing breath to start speaking, to explain just what they mean. If I did read out the slides no-one would be any the wiser which is at it should be. There are lots of pictures but they are all of the same sort of thing. A human brain can grasp the shared connection between the images in a second as it one of the key emotional triggers i.e. food. Another key emotional trigger is sex but that is best avoided in technical presentations otherwise no-one will listen to a word you say. Anyway, what I say to the audience is:-

As soon as you get back to your desk: -

1. Pick a candidate program – one that has been the bane of your life for years because it

keeps breaking every time you change it. The time has come to show it who is boss.

2. Explain to your colleagues the (not so) radical thing you are about to do.
3. Do it! For every change to that candidate program from now on. Instead of getting worse with every change it will get *better* and (*close so most important sentence*) with every month that goes by *you* can make changes faster with less risk.

Resources – not shown during presentation

https://blogs.sap.com/2013/04/18/are-you-writing-evil-abap-code/

https://www.amazon.com/Working-Effectively-Legacy-Michael-Feathers/dp/0131177052

https://blogs.sap.com/2018/04/07/actual-tdd-in-real-life/
https://blogs.sap.com/2018/02/03/back-to-the-future-reloaded-part-2/

https://blogs.sap.com/2018/03/28/open-sap-course-unit-testing-week-three/

https://github.com/epeterson320/ABAP-Logger

https://github.com/SAP/styleguides/blob/master/clean-abap/CleanABAP.md
https://github.com/SAP/code-pal-for-abap

Technical Note : A PowerPoint presentation should be very light on text – otherwise the audience will focus on reading that text and not listen to the speaker.

Thus I put mainly pictures in the slides, and have links (Mice with Placards) to show that on the last slide there are assorted reference material which covers the topic presented in much more detail, which the audience members can read at their leisure after the event.