COLLABORATE19
TECHNOLOGY AND APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

Don’t Discount the Developer

Tales from the Technical Dark Side

Remember to complete your evaluation for this session within the app!

Session ID:
11249

Prepared by:
Joe Tseng/Tammy Vandermey
O2Works LLC

Monday, April 8, 2019

Agenda

- Introductions
- Dark Side Technical Practices and Procedures
- Dark Side Coding Patterns
- Stepping Into the Light – Going Forward Recommendations

Introductions

- Joe Tseng
  - Technical EBS Consultant, O2Works, LLC
  - Over 25 years technical implementation experience in Oracle EBS
  - Contact Information: jtseng@o2works.com
- Tammy Vandermey
  - Technical EBS Consultant, O2Works, LLC
  - Over 25 years technical implementation experience in Oracle EBS
  - Contact Information: tammy@o2works.com

About O2Works

O2Works is one of the leading E-Business Suite services providers offering the most experienced teams of functional and technical consultants in the industry. Our hands-on ***resources average 20+ years of experience*** focused exclusively on implementing, upgrading, integrating, and extending Oracle's E-Business Suite. Stop by and talk to us about our large portfolio of successful projects.

ORACLE Gold Partner

O2Works *Putting Oracle To Work*

OAUG 2019 STAR PARTNER LEVEL THREE

Stop by and visit us at Booth 601 in the Exhibition Hall

Presentations, White Papers, and other information shared on-line at: https://o2works.com/knowledge-works/

COLLABORATE19
TECHNOLOGY AND APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

#C19TX

Dark Side Practices and Procedures

The Technical Dark Side
Practices and Procedures

- Based on experience, there are many technical practices that can lead a project or an IT group to the technical dark side. The ones we have seen often include the following:
  - Believe that any technical resource will do
  - Underestimate the Importance of Source Code Control
  - Overlook the Importance of Deployment Tools or Standards
  - Failure to Instance Plan
  - Documenting for the Sake of Documenting

Dark Side Practices and Procedures
"Any Resource Will Do"

- The proper selection of technical resources is an often overlooked aspect to any IT project. Failure to utilize competent resources can be catastrophic to project success.
- Technical resource selection is sometimes characterized by following beliefs:
    - Cheaper is better.
    - Off-shore means a 24 hour project cycle and better productivity
    - It's the engagement partners responsibility
    - Resumes are all that's needed, forget the interview
    - Module experience trumps long-term industry experience

Dark Side Practices and Procedures
"Any Resource Will Do"

- Buyer beware, these things actually do happen:
  - Engagement companies don't actually have known technical resources, but will find them anonymously off the "street" when projects arise.
  - Engagement companies send more experienced technical resources to interviews and swap them out later – after the opening stages of projects. These resources are sometimes replaced by resources with minimal work experience
  - Phone interviews are conducted by experienced professionals but different resources show up on site
  - Multiple resources are assigned to the same project tasks in order to "train" more resources.
  - Junior resources "work" during the day at client sites but turn over actual coding to more experienced off-shore resources in the off-hours.
  - Junior resources use client sites as opportunities to experiment and learn.

Dark Side Practices and Procedures
"Any Resource Will Do"

- The end result of poor technical resource choices
  - Last-minute, frantic go-lives that often fail to meet dates or deadlines
  - Project cost explosion
  - Costly Engagement partner replacement
  - Performance problems
  - Poor code that is difficult and costly to maintain
  - Un-orthodox coding methods that violate standards and thus may invalidate Oracle support agreements
  - Costly rewrites

Dark Side Practices and Procedures

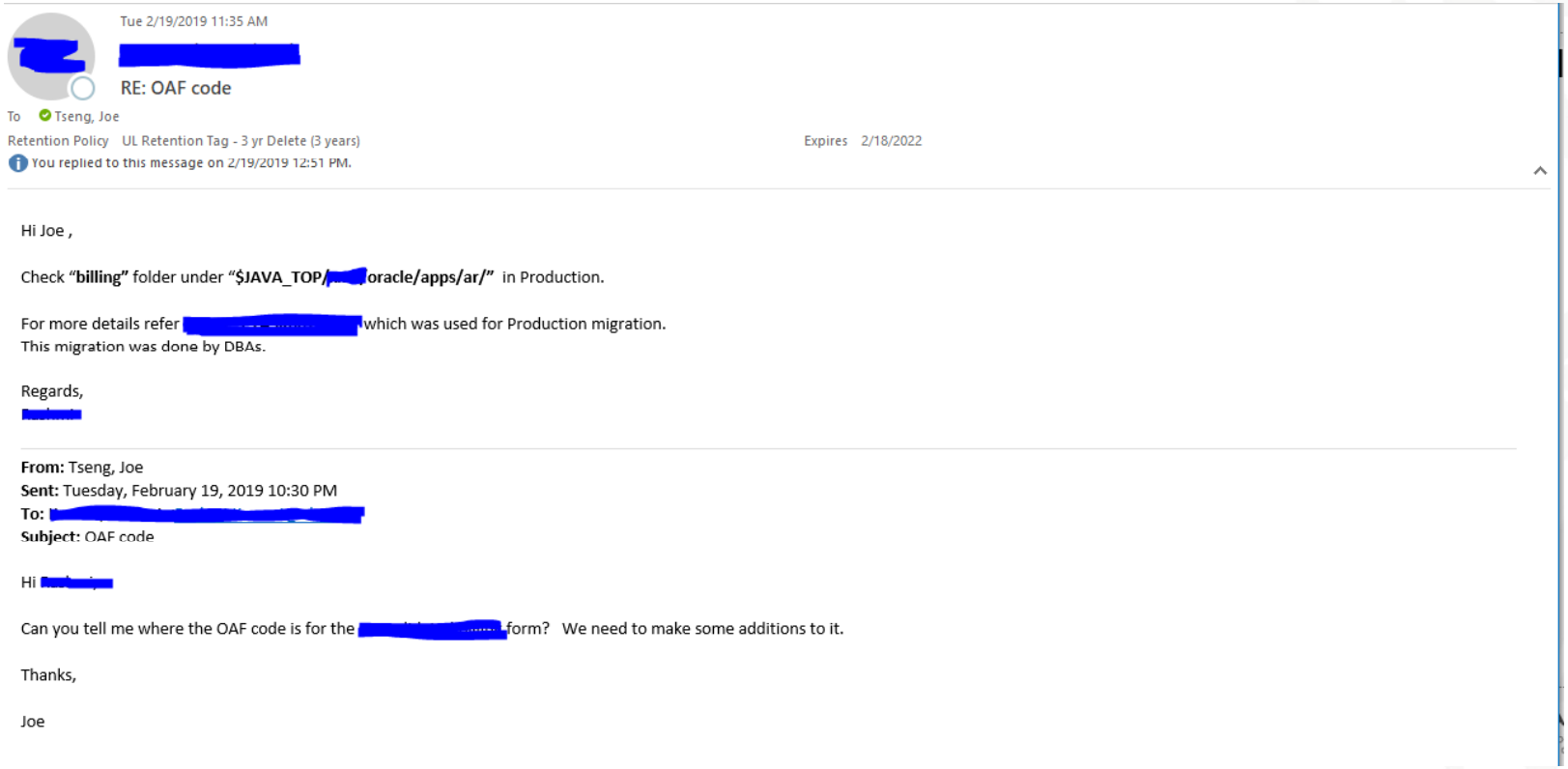
"Underestimate the Importance of Tools – Source Code Control"

- Every environment needs an active and usable source code control repository tool.
  - Creating date named folders in Microsoft Windows Explorer is NOT a tool
  - Using your Production environment for source code is NOT a tool

- Not having and using a source code control tool is an invitation to disaster
  - Lost, working versions of code can set a project timeline back as functionality is rebuilt
  - Accidental overwrites or loss of code

Dark Side Practices and Procedures

“Underestimate the Importance of Tools – Source Code Control”



Tue 2/19/2019 11:35 AM

[redacted]

RE: OAF code

To Tseng, Joe

Retention Policy UL Retention Tag - 3 yr Delete (3 years) Expires 2/18/2022

You replied to this message on 2/19/2019 12:51 PM.

Hi Joe ,

Check **“billing”** folder under **“$JAVA_TOP/[redacted]oracle/apps/ar/”** in Production.

For more details refer [redacted] which was used for Production migration.
This migration was done by DBAs.

Regards,
[redacted]

From: Tseng, Joe
Sent: Tuesday, February 19, 2019 10:30 PM
To: [redacted]
Subject: OAF code

Hi [redacted]

Can you tell me where the OAF code is for the [redacted] form? We need to make some additions to it.

Thanks,

Joe

Dark Side Practices and Procedures

"Underestimate the Importance of Tools – Source Code Control"

- There are plenty of tool options available for purchase.
- If you choose not to purchase a tool, there are plenty of free source code control repository tools available
  - SVN
  - PVCS
  - Git
  - Tortoise SVN (SVN w/ a Windows Shell)
  - Tortoise Git (Windows Shell for Git)

- Source Code Control is critical to maintaining a version history of all source code objects and being able to diagnose code control issues.

Dark Side Practices and Procedures

"Underestimate the Importance of Tools – Source Code Control"

- At many sites that have made the well intentioned investment into a source code control tool, the improper use of the tool is a point of failure
  - Code Dumping grounds – Failure to properly categorize code can lead to a massive object dumping ground – or even duplicate code within the tool repository
  - Incomplete code control – Critical objects may be missing from code control.
  - Lack of integration with deployment methodology or tool

Dark Side Practices and Procedures

"Overlook Deployment"

- An inadequate view of the importance of code deployment is another contributor to the technical dark side
  - A solution must not be viewed as "working" unless it is proven that it can be properly and consistently deployed to Production.
  - Proper testing includes deployment that follows an identical path to production
  - Deployments that have gone wrong can be costly
- Deployment Tools in an EBS environment can be expensive
  - May need custom tailoring to your environment
  - Have much thought and practical experience built into them
- The alternative to a deployment tool may be standard procedures and scripts
  - Require training specific to your environment
  - Often involve scripts that need to be suited to the specific solution
  - Can be error prone

Dark Side Practices and Procedures
"Overlook Deployment"

- At a minimum, a good Deployment Tool or Methodology will include the following:
  - Ability to integrate with source code control. Deployments must be able to pull specific versions of code
  - Ability to deploy to any EBS environment
  - Ability to deploy all types of EBS objects
    - Database objects (tables, views, PLSQL packages, etc)
    - Oracle Forms and Reports
    - BI Publisher templates and Data Templates
    - OAF pages
    - Configurations

Dark Side Practices and Procedures

"Failure to Instance Plan"

- Instance Planning is another overlooked aspect of the EBS technical picture
  - Having everyone "swim" in a 3 year old testing instance is not a plan
  - Stale instances lead to stale results
  - Instance refreshes must be integrated into a project plan.
  - Optimal environments have an instance cloned weekly, if not daily.
  - Investments in hardware, personnel, and procedures to utilize available cloning tools are well worth it.

Dark Side Practices and Procedures

"Documenting for the Sake of Documenting"

- Too often, IT environments often lack a well thought out plan for storing and cataloging documentation.
  - Documentation dumping grounds are the "norm". Approaches are often scattered and very often lack a means of finding proper documentation, thus further perpetuating the problem. These dumping grounds can be found in
    - Windows folders
    - SharePoint sites
    - Source Code Control
  - Additionally, the requirement to use outdated templates also results in the creation of documentation that do not adequately describe the solution being implemented.
  - Without a plan, the reality is that though a time consuming part of the project, created documentation is practically useless.

Dark Side Coding Practices

Dark Side Coding Practices

- Over the years, we've seen a number of coding practices that are less than optimal. Though invisible to most, poorly code written code is costly for a number of reasons
  - Code Maintenance - The inability to read code can be expensive as time is spent weeding through unnecessarily complex or bloated code
  - Code "inertia" – Functionality that is initially written with poor code must be replicated if a rewrite is necessary
  - Performance problems – Poor coding approaches and spaghetti logic often leads to poor performance
  - Violation of Oracle standards – Code that bypasses API's violates Oracle support agreements
  - Organizations can be held "hostage" by poor code
  - Rewrites of major functionality as code is finally deemed to be too unstable

Dark Side Coding Patterns

1. The Clown Car
2. The Superman
3. The Tree Killer
4. The 7-11 – The Database is not your convenient store
5. The Ted Kennedy
6. The Matryoshka Doll
7. The Move Along Nothing To See Here
8. The NULLIFIER
9. The Straight Jacket
10. The Hard Hat

Dark Side Coding Patterns

#1 – “The Clown Car”

- Failing to modularize your code, putting everything in a single code unit.

- Consequences:
  - No code re-usability - Single use code
  - Poor Code maintainability

Dark Side Coding Patterns

#2 – “The Superman”

- Instead of working out access issues properly, set context to be whatever works.

- Consequences:
  - Unintended operations are made with unintended code authorizations.

Dark Side Coding Patterns

#3 – “The Tree Killer”

- Instead of using records, declare variables for EVERYTHING.

- Consequences:
  - Unnecessary Code Bloating
  - Poor Code Maintainability

Dark Side Coding Patterns

#3 – “The Tree Killer”

```
l_return_status         VARCHAR2 (1) := 'S';
l_msg_count             NUMBER;
l_msg_data              VARCHAR2 (2000);
l_cust_trx_id           NUMBER;
l_cust_cm_id            NUMBER;
l_trx_number            VARCHAR2 (20);
l_customer_id           NUMBER;
l_invoice_number        VARCHAR2 (60);
l_invoiceid             VARCHAR2 (60);
l_billing_type          VARCHAR2 (256)   := NULL;
l_inv_amt               NUMBER;
l_rec_amt               NUMBER;
l_cm_amt                NUMBER;
l_ref_rec_amt           NUMBER;
l_ref_cm_amt            NUMBER;
l_applied_cm_amt        NUMBER;
l_applied_rec_amt       NUMBER;
l_qty                   NUMBER;
cnt                     NUMBER          := 0;
l_count                 NUMBER          := 0;
l_cash_receipt_id       NUMBER;
l_invoice_count         NUMBER          := 0;
l_cm_count              NUMBER          := 0;
l_msg_data_out          VARCHAR2 (240);
l_party_name            VARCHAR2 (100);
l_customer_trx_id       NUMBER;
l_trx_header_id         NUMBER;
l_invoice_error         VARCHAR2 (240);
l_receipt_method_id     NUMBER;
l_amount_applied        NUMBER;
l_currency              VARCHAR2 (20);
l_exchange_rate_type    VARCHAR2 (20);
l_exchange_rate_date    VARCHAR2 (20);
l_bill_to_site_use_id   NUMBER;
l_ship_to_site_use_id   NUMBER;
l_description           VARCHAR2 (240);
l_pmt_method            VARCHAR2 (20);
l_bank_account_id       NUMBER          := NULL;
l_bank_cnt              NUMBER;
l_party_id              NUMBER;
l_trx_type_id           NUMBER;
l_total_count           NUMBER          := 0;
l_line_count            NUMBER          := 0;
l_tot_line_count        NUMBER          := 0;
l_count_bad_cust        NUMBER          := 0;
l_total_errcount        NUMBER          := 0;
l_total_refund_usd      NUMBER          := 0;
l_usta_remit            VARCHAR2 (60);
l_remit_message         VARCHAR2 (1000);
l_source                VARCHAR2 (20);
l_trx_date              DATE;
l_bank_account_type     VARCHAR2 (20);
l_amt_applied_total     NUMBER          := 0;
l_div_id                VARCHAR2 (150);
l_privatelabelgroup     VARCHAR2 (80);
l_invoice_date          DATE;
l_days_late             NUMBER;
```

Dark Side Coding Patterns

#4 – “The 7-11”

- Using the database as a convenient store – visiting it as often as possible.

- Consequences
  - Poor program performance
  - Redundant logic
  - Poor Code maintainability

Dark Side Coding Patterns

#4 – “The 7-11”

```
CREATE OR REPLACE FORCE VIEW "APPS"."ARBPA_[redacted]EADER_V" ("INIT", "INTERCO_DATE", "BILLTO_COUNTRY", "SHIP_TO_COUNTRY", "CANADA_GST_TAX",
SELECT xxul_bnp_bpa_utils_pkg.init (rcta.customer_trx_id) init,[redacted]
        xxul_bnp_bpa_utils_pkg.get_interco_date (rcta.customer_trx_id)
           interco_date,
        xxul_bnp_bpa_utils_pkg.get_bill_to_country (rcta.customer_trx_id)
           billto_country,
        xxul_bnp_bpa_utils_pkg.get_ship_to_country (customer_trx_id)
           ship_to_country,
        xxul_bnp_bpa_utils_pkg.get_ca_gst_tax (customer_trx_id)
           canada_gst_tax,
        xxul_bnp_bpa_utils_pkg.get_ca_qst_tax (customer_trx_id)
           canada_qst_tax,
        xxul_bnp_bpa_utils_pkg.get_bill_to_state (customer_trx_id)
           bill_to_state,
        xxul_bnp_bpa_utils_pkg.get_ship_to_state (customer_trx_id)
           ship_to_state,
        xxul_bnp_bpa_utils_pkg.get_bill_to_province (customer_trx_id)
           bill_to_province,
        xxul_bnp_bpa_utils_pkg.get_ship_to_province (customer_trx_id)
           ship_to_province,
        xxul_bnp_bpa_utils_pkg.turkey_currency_switch (
           rcta.customer_trx_id)
           tur_currency,
        xxul_bnp_bpa_utils_pkg.get_turkish_conversion_rate (
           customer_trx_id)
           tur_conv,
        xxul_bnp_bpa_utils_pkg.get_fus_total_amt_due (rcta.customer_trx_id)
           fus_total_amt_due,
        xxul_bnp_bpa_utils_pkg.get_applicant_acct_num (
           rcta.customer_trx_id)
           applicant_acct_num,
        xxul_bnp_bpa_utils_pkg.get_applicant_addr (rcta.customer_trx_id)
           applicant_addr,
        xxul_bnp_bpa_utils_pkg.xxul_bnp_get_ca_qst_reg_num (
           rcta.customer_trx_id)
           qst_reg_num,
        xxul_bnp_bpa_utils_pkg.xxul_bnp_get_ca_gst_reg_num (
           rcta.customer_trx_id)
           gst_reg_num,
        xxul_bnp_bpa_utils_pkg.get_canada_gst_tax_type (
           rcta.customer_trx_id)
           canada_gst_tax_type,
        xxul_bnp_bpa_utils_pkg.get_canada_qst_tax_type (
           rcta.customer_trx_id)
           canada_qst_tax_type,
        xxul_bnp_bpa_utils_pkg.get_canada_gst_tax_rate (
           rcta.customer_trx_id)
           canada_gst_tax_rate,
        xxul_bnp_bpa_utils_pkg.get_canada_qst_tax_rate (
           rcta.customer_trx_id)
           canada_qst_tax_rate,
        xxul_bnp_bpa_utils_pkg.get_canada_gst_tax_amount (
           rcta.customer_trx_id)
           canada_gst_tax_amount,
        xxul_bnp_bpa_utils_pkg.get_canada_qst_tax_amount (
           rcta.customer_trx_id)
```

Dark Side Coding Patterns

#5 – “The Ted Kennedy”

- Left justify your code so its impossible to read

- Consequences:
  - Poor Code maintainability

Dark Side Coding Patterns

#5 – “The Ted Kennedy”

```
IF lv_item_number IS NOT NULL
THEN
BEGIN
SELECT instance_id
INTO ln_instance_id
FROM (SELECT cii.instance_id
FROM csi_item_instances cii,
mtl_system_items_b msib
WHERE 1 = 1
AND cii.active_end_date IS NULL
AND msib.organization_id =
13[redacted]
--
AND UPPER (msib.inventory_item_status_code) =
'ACTIVE'
AND cii.system_id IN (
SELECT system_id
FROM csi_systems_b
WHERE parent_system_id IN (
SELECT system_id
FROM (SELECT cii.instance_id,
cii.system_id,
cst.NAME
system_number
FROM apps.csi_item_instances cii,
apps.hz_cust_accounts hzc,
apps.csi_systems_tl cst
WHERE cii.owner_party_account_id =
hzc.cust_account_id
AND hzc.account_number =
lv_account_num
AND hzc.status =
'A'
AND cii.install_location_id =
lv_install_site
AND cst.system_id =
cii.system_id
AND cst.LANGUAGE =
USERENV
('lang')
AND cii.active_end_date IS NULL
AND instance_type_code =
(SELECT lookup_code
FROM apps.csi_lookups l
WHERE l.lookup_type(+) =
'CSI_INST_TYPE_CODE'
AND UPPER
(l.meaning
) =
UPPER
('MANUFACTURER'
)))))
AND cii.inventory_item_id =
msib.inventory_item_id
AND msib.segment1 =
NVL (lv_item_number, msib.segment1)
```

Dark Side Coding Patterns

#6 – “The Matryoshka Doll”

- Instead of modularizing, nest your IF conditions and blocks to a ridiculous depth.

- Consequences:
  - Poor Code Maintainability
  - Redundant Logic
  - Loss of modularity and re-use

Dark Side Coding Patterns

#6 – “The Matryoshka Doll”

```
IF cuv_applicant_create.US_ETHNIC_ORIGIN = 'Hispanic or Latino' THEN x_pei_information1 :='Y';
    ELSE
        IF cuv_applicant_create.US_ETHNIC_ORIGIN = 'American Indian or Alaska Native (Not Hispanic or Latino)' THEN x_pei_information2 :='Y';
            ELSE
                IF cuv_applicant_create.US_ETHNIC_ORIGIN = 'Asian (Not Hispanic or Latino)' THEN x_pei_information3 :='Y'   ;
                    ELSE
                        IF cuv_applicant_create.US_ETHNIC_ORIGIN = 'Black or African American (Not Hispanic or Latino)' THEN x_pei_information4 :='Y';
                            ELSE
                                IF cuv_applicant_create.US_ETHNIC_ORIGIN = 'Native Hawaiian/Other Pacific Islander(Not Hispanic/Latino)' THEN x_pei_information5 :='Y' ;
                                    ELSE
                                        IF cuv_applicant_create.US_ETHNIC_ORIGIN = 'White (Not Hispanic or Latino)' THEN x_pei_information6 :='Y'  ;
                                            ELSE
                                                IF cuv_applicant_create.US_ETHNIC_ORIGIN = 'Two or More Races (Not Hispanic or Latino)' THEN x_pei_information7 :='Y';
                                                    ELSE
                                                        IF cuv_applicant_create.US_ETHNIC_ORIGIN = 'Opt Out' THEN x_pei_information8 := 'Opt Out';
                                                        END IF;
                                                END IF;
                                        END IF;
                                END IF;
                        END IF;
                END IF;
        END IF;
END IF;
```

Dark Side Coding Patterns

#7 – “The Move-Along-Nothing-To-See-Here”

- Hide Exceptions so no one knows or log them to a place no one will ever look. This is otherwise known as the infamous “WHEN OTHERS THEN NULL”

- Consequences
  - Lost errors that can have unknown multiple downstream ramifications

Dark Side Coding Patterns

#8 – “The NULLIFIER”

- Instead of understanding and using variable scope within block structures, set all your variables to NULL “JUST IN CASE”

- Consequences:
  - Poor Code Maintainability
  - Possibility of unintended consequences when failing to reset variables

Dark Side Coding Patterns

#8 – “The NULLIFIER”

```
ln_price := NULL;
lv_salesper := NULL;
ln_gl_id_rev := NULL;
ln_location := NULL;
ln_gl_default_seg := NULL;
lv_segment := NULL;
lv_table_name := NULL;
ln_seg_num := NULL;
lv_segment1 := NULL;
lv_segment2 := NULL;
lv_segment3 := NULL;
lv_segment4 := NULL;
lv_segment8 := NULL;
lv_segment9 := NULL;
lv_trnsegment1 := NULL;
lv_trnsegment2 := NULL;
lv_trnsegment3 := NULL;
lv_trnsegment4 := NULL;
lv_segment5 := NULL;
lv_segment6 := NULL;
lv_segment7 := NULL;
lv_trnsegment8 := NULL;
lv_trnsegment9 := NULL;
---sales
lv_salesegment1 := NULL;
lv_salesegment2 := NULL;
lv_salesegment3 := NULL;
lv_salesegment4 := NULL;
lv_salesegment8 := NULL;
lv_salesegment9 := NULL;
---service
lv_sersegment1 := NULL;
lv_sersegment2 := NULL;
lv_sersegment3 := NULL;
lv_sersegment4 := NULL;
lv_sersegment8 := NULL;
lv_sersegment9 := NULL;
lv_default := NULL;
--
lv_tbl_seg1 := NULL;
lv_tbl_seg2 := NULL;
lv_tbl_seg3 := NULL;
lv_tbl_seg4 := NULL;
lv_tbl_seg8 := NULL;
lv_tbl_seg9 := NULL;
in_ln_org_id := NULL;
---Version 1.1 Starts
ln_contract_number := NULL;
ln_line_number := NULL;
```

Dark Side Coding Patterns

#9 – “The Straight Jacket”

- Limit your code to just 80 characters and wrap as needed

- Consequences:
  - Poor Code Maintainability and Readability

Top Ten Coding Pattern Killers

#9 – “The Straight Jacket”

```
      IF in_psn IS NULL
      THEN
         o_error_msg :=
            'PSN is required field for Service Contracts/IB details service';
         o_error_code := 2;
      ELSE
         FOR srv_hdr_dtls_rec IN srv_hdr_dtls_cur
         LOOP
            out_sc_contract_hdr_tbl.EXTEND;
            out_sc_contract_hrd_rec.service_contract_number :=
                                        srv_hdr_dtls_rec.contract_number;
            out_sc_contract_hrd_rec.contract_header_id := srv_hdr_dtls_rec.ID;
            out_sc_contract_hrd_rec.contract_number_modifier :=
                                  srv_hdr_dtls_rec.contract_number_modifier;
            out_sc_contract_hrd_rec.description :=
                                       srv_hdr_dtls_rec.short_description;
            out_sc_contract_hrd_rec.contract_start_date :=
                                     srv_hdr_dtls_rec.contract_start_date;
            out_sc_contract_hrd_rec.contract_end_date :=
                                       srv_hdr_dtls_rec.contract_end_date;
            out_sc_contract_hrd_rec.total_amount :=
                                          srv_hdr_dtls_rec.total_amount;
            out_sc_contract_hrd_rec.negotiation_status :=
                                    srv_hdr_dtls_rec.negotiation_status;
            out_sc_contract_hrd_rec.contract_group := srv_hdr_dtls_rec.NAME;
            out_sc_contract_hrd_rec.status := srv_hdr_dtls_rec.sts_code;
            out_sc_contract_hrd_rec.cust_po_number :=
                                          srv_hdr_dtls_rec.cust_po_number;
            out_sc_contract_hrd_rec.payment_term :=
                                          srv_hdr_dtls_rec.payment_term;
            out_sc_contract_hrd_rec.price_list := srv_hdr_dtls_rec.price_list;
            out_sc_contract_hrd_rec.customer_party_id :=
                                           srv_hdr_dtls_rec.party_id;
            out_sc_contract_hrd_rec.customer_party_number :=
                                          srv_hdr_dtls_rec.party_number;
            out_sc_contract_hrd_rec.party_site_number :=
                                       srv_hdr_dtls_rec.party_site_number;
            out_sc_contract_hrd_rec.customer_name :=
                                           srv_hdr_dtls_rec.party_name;
            out_sc_contract_hrd_rec.cust_account_number :=
                                         srv_hdr_dtls_rec.account_number;
            out_sc_contract_hrd_rec.cust_account_id :=
                                       srv_hdr_dtls_rec.cust_account_id;
            --  Get bill to account details
            get_bill_to_details
                        (srv_hdr_dtls_rec.bill_to_site_use_id,
                         out_sc_contract_hrd_rec.bill_to_cust_party_number,
                         out_sc_contract_hrd_rec.bill_to_cust_party_name,
                         out_sc_contract_hrd_rec.bill_to_party_acc_number,
                         out_sc_contract_hrd_rec.bill_to_party_site_number,
                         out_sc_contract_hrd_rec.bill_to_party_location,
                         out_sc_contract_hrd_rec.bill_to_cust_address_line1,
                         out_sc_contract_hrd_rec.bill_to_cust_address_line2,
                         out_sc_contract_hrd_rec.bill_to_cust_address_line3,
                         out_sc_contract_hrd_rec.bill_to_cust_address_line4,
                         out_sc_contract_hrd_rec.bill_to_cust_city
```

Top Ten Coding Pattern Killers

#10 – “The Hard Hat”

- Instead of parameterizing, hard code literal values whenever you can

- Consequences:
  - Poor Code Maintainability
  - Loss of code re-use

Top Ten Coding Pattern Killers

#10 – “The Hard Hat”

```
-- Initialize Item Details for UL Mark
SELECT segment1,
       description
  INTO g_ys_service_items('ULMARK').item_number,
       g_ys_service_items('ULMARK').item_description
FROM mtl_system_items_b
WHERE organization_id = 133
AND segment1 = '30201237';

-- Initialize Item Details for ALSP
SELECT segment1,
       description
  INTO g_ys_service_items('ALSP').item_number,
       g_ys_service_items('ALSP').item_description
FROM mtl_system_items_b
WHERE organization_id = 133
AND segment1 = '30201236';

-- Initialize Item Details for ALSP Cover Line
SELECT segment1,
       description
  INTO g_ys_service_items('ALSPCOV').item_number,
       g_ys_service_items('ALSPCOV').item_description
FROM mtl_system_items_b
WHERE organization_id = 133
AND segment1 = '30034437';

-- Initialize Item Details for Inspection Service
SELECT segment1,
       description
  INTO g_ys_service_items('SRV').item_number,
       g_ys_service_items('SRV').item_description
FROM mtl_system_items_b
WHERE organization_id = 133
AND segment1 = '30014613';

-- Initialize Item Details for Production Volume
SELECT segment1,
       description
  INTO g_ys_service_items('PV').item_number,
       g_ys_service_items('PV').item_description
FROM mtl_system_items_b
WHERE organization_id = 133
AND segment1 = '30027940';

-- Initialize Item Details for Annual Fee
SELECT segment1,
       description
  INTO g_ys_service_items('AF').item_number,
       g_ys_service_items('AF').item_description
FROM mtl_system_items_b
WHERE organization_id = 133
AND segment1 = '30027937';
```

COLLABORATE19
TECHNOLOGY AND APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

#C19TX

Stepping into the Light

Going forward Recommendations

Stepping Into the Light
Going forward Recommendations

- Interview your technical resources before bringing them onboard
  - Request code samples to review coding techniques. Coding styles obviously vary between developers, but good coding practices will always include the following:
    - Well structured, intentional and consistent indentation scheme
    - Modularization and parameterization for reusability and maintainability
    - Limited code unit size
    - Generous use of free space to improve readability
    - Use of block structures to properly assign variable scope and life
    - Code that is self-documenting
      - Use of sensible variable names
      - Use of Anchored Data types as a means of documenting code intent
      - Use of table and cursor record types
      - Avoidance of redundant comments
  - Have resources write actual code **in person** as part of the vetting process
  - Use interview question that are scenario based that will identify actual problem solving skills

Stepping Into the Light
Going forward Recommendations

- If using an engagement partner, ask questions about who they have assigned to your project
  - How long have they been with the company
  - Ask for references
- Understand that despite the marketing material, off-shore technical work is not necessarily cheaper
- Choose experience over “exposure” to a specific needed module
- Use experienced resources for more complex tasks

Stepping Into the Light
Going forward Recommendations

- Implement a Source Code Control tool NOW if you have not yet done so
  - Make an investment into studying how best to structure the repository to suit your needs
  - Integrate your source code control tool with your deployment tool or methodology
- Do not overlook the importance of a clean solution deployment
- Review instance planning within your organization or project
- Re-think the why and how of technical documentation
- Perform code reviews as necessary to avoid bad coding patterns

COLLABORATE19

TECHNOLOGY AND APPLICATIONS FORUM FOR THE ORACLE COMMUNITY

Q & A

Joe Tseng – jtseng@o2works.com

Tammy Vandermey – tammy@o2works.com

Session ID:

11249

Remember to complete your evaluation for this session within the app!