Stay Out of APEX Debt

Scott Spendolini
Vice President, APEX+ Practice

VISCOSITY
NORTH AMERICA

Welcome

About Me

scott.spendolini@viscosityna.com
@sspendol

About Viscosity: Services

DATA	APPS	INFRA
Database	DevOps Microservices	Public Cloud Bare Metal
Data Warehouse Analytics	Java & APEX	Engineered Systems
Data Replication & Virtualization	PaaS for SaaS	x86/SPARC
OCI & AWS	Systems Integration	OCI AWS Bare Metal
	POCs	

About Viscosity: APEX

- Offers a wide range of APEX related Services:
  - Architecture Design & Reviews
  - Security Reviews
  - Health Checks
  - Education
    - On-site, On-line, On-Demand
    - Custom & Mentoring
  - DevOps
  - Curators of APEX-SERT & sumnerAMP

ORACLE Platinum Partner

Agenda

- Overview
- Technical Debt in APEX
  - Applications
  - Infrastructure
- Summary

Memes

- Are (usually) funny
- Are (almost always) true
- Which serious issue so that th g that directl

Overview

Technical Debt

Technical Debt

- **Technical debt** is a concept in software development that reflects the **implied cost of additional rework** caused by **choosing an easy solution now** instead of **using a better approach that would take longer**.
- Technical debt can be **compared to monetary debt**. If technical debt is not repaid, **it can accumulate 'interest'**, making it harder to implement changes later on.

https://en.wikipedia.org/wiki/Technical_debt

Causes of Technical Debt

- Insufficient up-front definition
- Business pressures
- Lack of process or understanding
- Tightly-coupled components
- Lack of a test suite
- Lack of documentation
- Lack of collaboration
- Delayed refactoring
- Lack of alignment to standards
- Lack of knowledge
- Lack of ownership
- Poor technological leadership
- Last minute specification changes

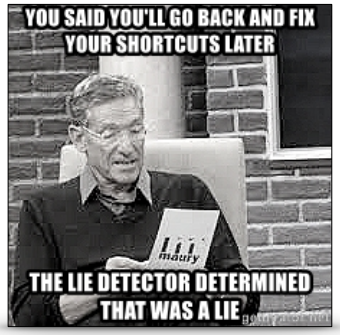
https://en.wikipedia.org/wiki/Technical_debt

Technical Debt

Lost Time

- Lost Time causes developers to:
  - Get stressed out
  - Work longer hours
  - Make more mistakes
  - **Take more shortcuts**

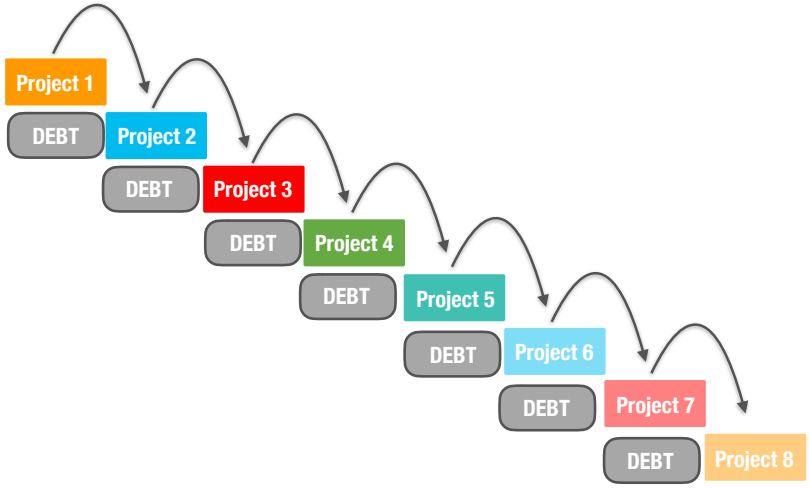
"I know this is a bad idea, but I'll fix it later."



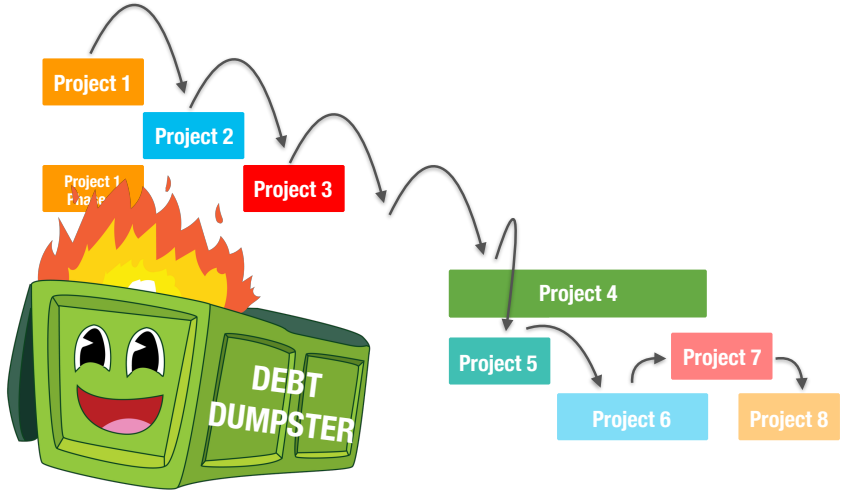
Trade Offs

- "**Shortcuts**" and hard-coded "**workarounds**" are good examples of technical debt
  - **You're trading a robust solution for a cheaper one and getting back some time**
  - Time that you can use for:
    - Sleep
    - Family
    - More Work
- The **cheaper, less optimal solution is the debt**
  - You accumulate debt as projects roll on
  - It doesn't just go away like in the real world

Technical Debt



Technical Debt



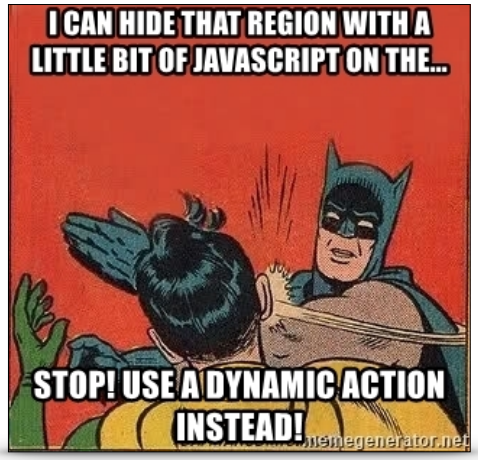
Technical Debt in APEX

Technical Debt in APEX

- Applications
  - JavaScript
  - SQL & PL/SQL
  - Plugins
  - Data Model
  - Documentation
  - Components
  - User Interface
- Infrastructure
  - New Versions
  - Patches
  - End Users

Applications

Javascript



Javascript

- How much **Javascript** should you include in your APEX application?
  - **As little as possible**
  - And when you do, it **should always be in a Dynamic Action**

Dynamic Actions

- Dynamic Actions are **declarative constructs** that allow developers to easily **manipulate client-side operations**
  - Change a value of a select list
    - If **A**, then **show Item A** and **hide Item B**
    - If **B**, then **show Item B** and **hide Item A**

Dynamic Actions

- APEX uses the **declarative values** of a Dynamic Action to automatically **generate jQuery code**

```
apex.da.initDaEventList = function(){

apex.da.gEventList = [

{"triggeringElementType":"ITEM","triggeringElement":"P1_SELECT","conditionElement":"P1_SELECT"
,"triggeringConditionType":"EQUALS","triggeringExpression":"A","bindType":"bind","bindEventTyp
e":"change","anyActionsFireOnInit":true,actionList:
[{"eventResult":false,"executeOnPageInit":true,"stopExecutionOnError":true,"affectedElementsTy
pe":"ITEM","affectedElements":"P1_B",javascriptFunction:apex.da.show,"attribute01":"N","action
":"NATIVE_SHOW"},
{"eventResult":true,"executeOnPageInit":true,"stopExecutionOnError":true,"affectedElementsType
":"ITEM","affectedElements":"P1_A",javascriptFunction:apex.da.show,"attribute01":"N","action":
"NATIVE_SHOW"},
{"eventResult":false,"executeOnPageInit":true,"stopExecutionOnError":true,"affectedElementsTyp
e":"ITEM","affectedElements":"P1_A",javascriptFunction:apex.da.hide,"attribute01":"N","action"
:"NATIVE_HIDE"},
{"eventResult":true,"executeOnPageInit":true,"stopExecutionOnError":true,"affectedElementsType
":"ITEM","affectedElements":"P1_B",javascriptFunction:apex.da.hide,"attribute01":"N","action":
"NATIVE_HIDE"}]}];

}
```


Dynamic Actions

- Since APEX generates the actual JavaScript, **if the underlying Javascript libraries were to change** - version or library itself - then **the way the Javascript is generated will also change**
  - Oracle will do this for you
  - You have to do nothing
  - You have to know nothing
  - Simply upgrade and enjoy your debt-free life

Javascript in DAs

- Sometimes, **you have no choice**, and **need to use Javascript**
- For instance
  - Execute PL/SQL asynchronously
  - Create an expression
- Any Javascript call should **always originate from a DA**
  - If you need to listen for item or button clicks, use the corresponding **id** or **class** as a trigger

Javascript APIs

- If this is the case, **always and only use the APEX Javascript APIs**
  - https://docs.oracle.com/en/database/oracle/application-express/18.2/aexjs/toc.html
  - Also available as part of the Oracle APEX API Documentation
- Just like DAs, APEX Javascript APIs **encapsulate logic** and **insulate you from change**
  - If jQuery syntax changes, API will accommodate it
  - Again, you'll need to do nothing

Javascript APIs

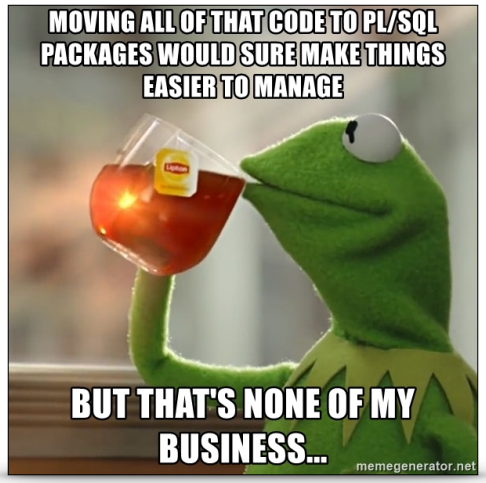
- Consider the case of **gReport**
  - From Joel Kallman's blog discussing APEX 5.0:

> As Trent Schafer (@trentschafer) noted in his latest blog post, "Reset an Interactive Report (IR)", there have been numerous customer discussions and blog posts which show how to directly use the gReport JavaScript object to manipulate an Interactive Report. The problem? **With the massive rewrite to support multiple Interactive Reports in Oracle Application Express 5, gReport no longer exists**. And as Trent astutely points out, **gReport isn't documented**. And that's the cautionary tale here - **if it's not documented, it's not considered supported or available for use and is subject to change, effectively without notice**. While I appreciate the inventiveness of others to do amazing things in their applications, and share that knowledge with the Oracle APEX community, **you must be cautious in what you adopt**.
>
> *https://joelkallman.blogspot.com/2015/02/some-changes-to-be-aware-of-as-oracle.html*

Deprecated & Desupported

- If for some reason, you're still using deprecated and/or desupported Javascript libraries, you can include them
  - Shared Components > UI Attributes > Desktop
  - Check either **Pre-18.1** or **18.x**
  - Specifics of what is deprecated and desupported is always included in the release notes of each APEX release

SQL & PL/SQL



Views

- Most - if not all - **source for APEX reports** should use **views**
  - Exception for single-table queries
- Complex join & other logic should be encapsulated in the view
  - Makes management much easier - change the view logic, all APEX reports are impacted
  - Add/Remove column - you're going to have to change APEX either way
- Views should also be used for data security
  - VPD or incorporate security context in the view

PL/SQL

- APEX processes allow you to put **blocks of PL/SQL in your application**
  - These blocks can be called from a variety of points on the page
  - Keep in mind this code is stored in the database as a CLOB
- Each time a process runs, it needs to be **fetched**, **parsed** and then **executed**
  - Which can get expensive if there's more then a few lines of code

PL/SQL

- Thus, all PL/SQL blocks **should be moved to a PL/SQL package** and referenced from APEX
  - Even small, one line blocks - such as **RETURN FALSE** - should be moved to a package
  - If the logic changes at some point, no change needs to be made to the application
- Managing PL/SQL via files is a **lot easier** than managing APEX application exports
  - Easier to diff, share, update, read, version control, etc.

PL/SQL

- Beware of using any **APEX-specific APIs** in PL/SQL packages
  - As soon as you do, that package can only be used from APEX
- If you have the need to call that package outside of APEX, then:
  - Pass in all parameters - do not use the "v" function
  - Create a wrapper procedure that does the "APEX Stuff"
- It's **perfectly OK** to have **APEX-specific packages**
  - Hard to avoid when using things like **APEX_APPLICATION.G_F01**, **APEX_COLLECTION**, etc.

PL/SQL APIs

- APEX contains a **robust set of PL/SQL APIs**
  - Included in the documentation
  - Some can even be called from outside of APEX
- These should be **studied and used whenever possible**
  - Your problem - although interesting - was likely solved years ago and is now incorporated into an API
- "Knowing is Half the Battle"
  - You don't know what you don't know
    - A stitch in time saves nine
      - You've got to know when to hold 'em…

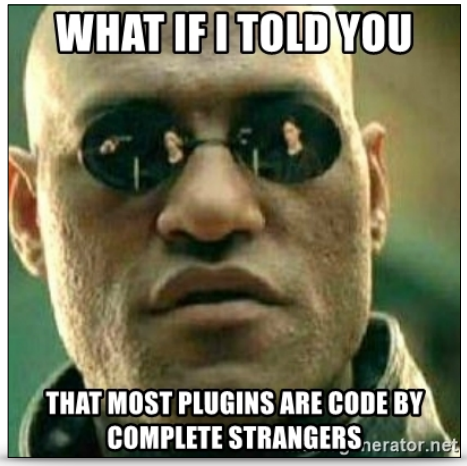
Migrating

- You can use **SQL Developer** to "migrate" PL/SQL processes from APEX to a package
  - Expand the APEX node
  - Right-click the application
  - Select "migrate"
- You will get a **single package** with all of the code encapsulated in it
  - Up to you to manually copy & paste the stubs back in each process
- Lesson here: **be proactive**

APEX DML Processes

- Final note: if you can make use of an **APEX DML Process**, then do so
  - Automatically generated when creating a form
  - Declarative way to perform DML
  - Automatically has lost update detection
  - Anyone can understand the logic

Plugins



Plugins

- Plugins are a very cool way to **expand the capabilities of APEX**
  - Introduced in APEX 4
  - All APEX components - reports, charts, items, etc. - are actually plugins
- **Many people will utilize Plugins** in their applications
- **Few will actually create Plugins**
  - Difficult to do because of the broad skill set requirements
    - APEX, Javascript, CSS, PL/SQL, SQL, etc.

Plugins

- Let's consider **SuperLOV**
  - Very popular plugin released in 2010
    - About 10k downloads to date
  - Revised regularly until Dan McGhan joined Oracle in 2013 or so
    - Regular Updates ceased due to Dan's new job
    - Others have since taken over development
    - Currently no version for APEX 18.2

Plugin Issues

- **Developers may not release updated versions at the same time APEX does**
  - Almost impossible to do, since APEX does not have a public beta program
- **APEX upgrades may break or disable the plugin**
  - See the first point
- **Developers for one reason or another, may abandon the plugin entirely**
  - Leaving you or the community to support it

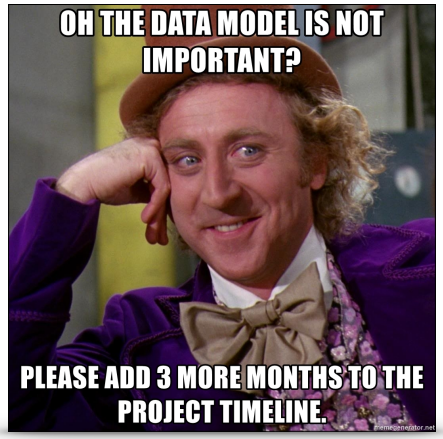
Plugin Considerations

- In the case of a plugin no longer working, you have two choices
  - **Fix it**
    - Difficult in many cases
  - **Replace it with native functionality**
    - Should have an existing plan for this
- **Most difficult choice will be to forego APEX updates b/c of plugin dependencies**
  - Sacrificing security, functionality, stability for the sake of what a plugin provides

jQuery Migrate

- If you're going to upgrade and do have plugs that are not working correctly, consider enabling **jQuery Migrate**
  - Shared Components > UI Attributes > Desktop
- Including jQuery Migrate **restores deprecated features and behaviors of jQuery** so that old JavaScript code and jQuery plug-ins will still run properly with the jQuery version loaded by Application Express.

Data Model



Data Model

- Like the **foundation of a house**, a **solid data model** is **essential** for any well-built application
  - Cutting corners may work in the short term
  - Important to know how big the house could be up front
    - Plan for expansion - even if it's unlikely
- Sometimes, you'll inherit someone else's mess
  - Be sure to take the time to rehabilitate it before starting anything new

Natural vs. Surrogate Keys

- APEX prefers **numeric surrogate keys automatically generated via trigger & sequence**
  - 12c Identity columns are OK too
- APEX doesn't care if you **disagree** with this statement
  - It still works best when you adhere to it
  - It will work if you don't - just not as optimally
- **GUIDs** will work, too
  - Harder to work with
  - Better if data from multiple systems or tables will be merged, since data is globally unique

Constraints

- All constraints should be at the **database level**
  - This way, the same business rules are enforced regardless of the technology
- APEX will automatically set "**Item Requires Value**" when a NOT NULL constraint is on a column
- APEX will **not interpret check constraints**
  - Up to the developer to handle these manually in APEX

Table APIs

- Consider creating **Table APIs**
  - API to facilitate DML transactions
- **SQL Workshop** can do this quickly
  - Object Browser > Create > Package > Package with methods on database table(s)
- Once created, you can **modify the code** to be less robust
  - Remove Insert or specific columns, for instance
- Simple to build a **RESTful interface** in front of these so that other technologies can use the same API

Documentation

Documentation

- Documenting an APEX application should be done in several different places
  - **Component Comments**
  - **Code Comments**
  - **Technical Documentation**
  - **User Documentation**

Component Comments

- Each and every APEX component contains a **Comments** field
- **Make sure to use it**
  - This may be the only documentation that you have
- **Bonus**: you can query this field from the APEX views
  - Or build an APEX application that can explore all comments

Code Comments

- **Comment your code**. Seriously.
  - I can't remember what I did this morning, let alone 2-3 months ago
- Comments should be **useful / meaningful**
  - More than just a restatement of the name of the procedure
  - Also helpful to include a log of major changes
  - Version control should do this, but it's a spot easier to have comments right in the package

Comment Examples

Bad

```
/* procedure to calculate
interest rate */

PROCEDURE calc_payment
  (p_rate IN NUMBER,
  p_terms IN NUMBER
)
RETURN NUMBER
IS
…
END calc_interest;
```

Good

```
/* calculates interest
based on amount and terms
in months, looks up
interest rate in INT_RATE
table, and returns a
monthly payment amount */

PROCEDURE calc_payment
  (p_rate IN NUMBER,
  p_terms IN NUMBER
)
RETURN NUMBER
IS
…
END calc_interest;
```

Technical Documentation

- This is for the **developers**
  - End users should and will never see this
  - So make it count
- Document any **transaction or flow**
  - Be sure to call out specific pages, views & PL/SQL packages used
- Most likely this will be **used in an emergency**
  - So keep it organized, up to date and succinct

User Documentation

- **No one writes documentation**, so don't feel bad about it
  - Seriously. You're not alone.
  - You're wrong, but not alone.

Components

Components

- Use APEX **components** as much as possible
  - Do not try to re-invent the wheel
- And more importantly, use them the **way they were intended** to be used
  - Using a component improperly can lead to security issues as well as maintenance nightmares
- If needed, **adapt your requirements** to meet the specification of APEX components

Components

- Examples of all components:
  - https://apex.oracle.com/pls/apex/f?p=42:3000
- If you can't find something that works from that selection, then **requirements need to be re-thought** so that you can
  - This is one of the largest ways to accumulate debt
  - A developer is forced by end users to make bad choices, customize a component outside the intended scope, and there's your debt

Reports

- **Queries**
  - Use Views where possible
  - Column Names should be defined in APEX
  - HTML does not belong in SQL - use column attributes
- **Attributes**
  - Standardize on template choice & attributes
  - Consider limiting which options of an IR are available

Forms

- Use the **built in DML processes** wherever and whenever possible
  - Completely declarative
  - No code required
  - Automatic lost update detection
- If not possible, consider using **table APIs**
  - Low Code
  - Logic can be centralized in a package and re-used, if needed

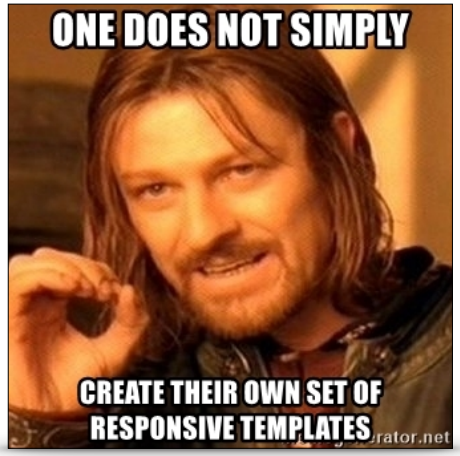
Items

- Use default naming standard: **PX_COL_NAME**
  - PX = P + Page Number
  - COL_NAME = corresponding database column
- Use the **grid layout controls** to arrange items
  - Columns
  - Column Span
  - New Line
- Application Items should have a **standard name**
  - AI_ITEM_NAME
  - G_ITEM_NAME

Shared Components

- Create a **page-less application** to stored common **shared components**
  - Authentication Schemes
  - Authorization Schemes
  - Lists of Values
  - Plugins
- When any of these components are needed, add to the central application and create a subscription
  - If anything needs to be changed, easier to do once and push vs. multiple times in multiple places

User Interface



Universal Theme

- Introduced in APEX 5, the **Universal Theme** is the new declarative construct to **manage an application's appearance**
  - There are no more "themes"
  - Universal Theme may as well just be called User Interface

Migrating to Universal Theme

- If you're using an older numbered theme, it's time to upgrade
  - All other themes are **deprecated in 18.1** and **desupported in 18.2**
- The more **HTML & CSS** you've added, the more difficult the transition will be
  - Hint: you're going to want to strip it down and use Theme Roller and/or new CSS

Migration

- Oracle offers a Universal Theme migration guide here:
  - https://apex.oracle.com/pls/apex/f?p=42:2000
  - Also part of **the Universal Theme Packaged Application**
- To simplify it, it's basically four steps:
  - **Backup your application**
  - Add the Universal Theme to your application
  - Switch to the Universal Theme
  - Clean up the mess

The Mess

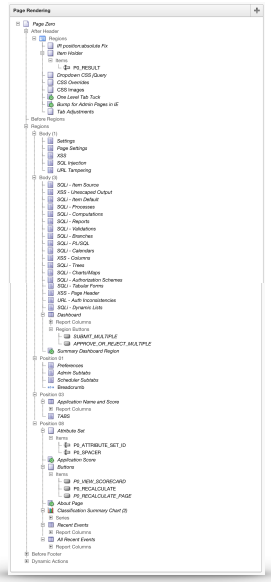
- **Navigation Menu**
  - May need to create or transition from tabs
- **Navigation Bar**
  - May not be a list
- **Two Levels of Tabs**
  - Need to convert to one level first
- **All Content in One Column**
  - Change new row/new column attributes
  - "Label Column Span" error

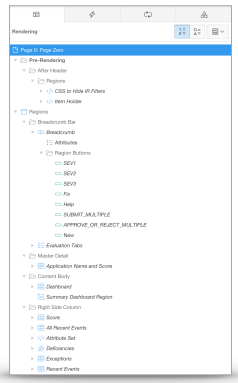
Your Mess

- Most, if not all of your customizations, **should be discarded**
  - Often an emotional decision, as they represent work and accomplishment
  - They will only bog down the Universal Theme
- You can **re-apply what you need** via Template Options and CSS overrides
  - Much more manageable
  - Easier to do once you get the hang of it

Case Study: APEX-SERT

- APEX-SERT was an APEX 4.x application with a 100% custom theme
  - Several CSS override regions on page 0
  - Many regions had **<style>** tags embedded
    - In a variety of places
  - Lots of manually generated HTML via PL/SQL
  - Excessively large number of Global Page components





Standards

- When migrating to the Universal Theme, it's critical to define **component standards**
  - Which page positions to use for what
  - When to use a Modal window and how they behave
  - Templates & Template Options for regions, buttons, etc.
  - Item Label Position & Options
- **Enforce standards** by using the **APEX views**
  - Examine each application to ensure that they comply

Stay Inside the Box

- Design to the UT's **supported structures & design patterns**
  - If your design does not fit, change it
- **Do not modify UT templates**
  - Make fixes via attributes or CSS
- Use **Template Options** and **CSS Overrides**
  - Often you can achieve what you're after with them and perhaps a small CSS tweak
    - Which should be done in Theme Roller

Universal Theme Rules

- Thou **shall not** edit the HTML in the templates
- Thou **shall not** use inline styles anywhere
- Thou **shall** use Theme Roller to modify basic styles
- Thou **shall** use Theme Options
- Thou **shall** test thy application in the target resolution of thy users
- Thou **shall** steer users towards supported design patterns and components and smite them should they digress

Infrastructure

New Versions

New Versions

- APEX has adopted the **same release cycle** as the rest of the database group
  - Smaller, more frequent releases with fewer features
  - APEX is on pace for about 2 per year
    - Which is way more than the previous pace of 1 per 2+ years
- Releases will be named in the following format:
  - **YY.VER**
    - YY = year
    - VER = release number

New Versions

- In addition to new features, each new release also contains:
  - **Bug fixes**
  - **Security patches**
- Thus, it's critical to **keep current** to be both more functional and more secure
- Adhering to all of the recommendations in this presentation will **minimize upgrade issues**
  - Enabling you to keep more current with fewer issues

Release Notes

- Critical to **read the Release Notes**
  - Shipped as part of the APEX documentation
- Important components:
  - Changed Behavior
  - Deprecated Features
  - Desupported Features

Changed Behavior

- Outlines **things that behave differently** in the current APEX release
  - Should read and determine if any changes will impact your environment
  - Some line items here will be new features/functionality

Deprecated Features

- Deprecated features are **features which Oracle plans to desupport or remove in a future release** of Oracle Application Express.
  - If a feature is related to application metadata or an API, existing applications can still use the feature, but Oracle strongly recommends that developers start to modify their applications as described in this section.
  - Use Oracle Application Express Advisor to scan existing applications for deprecated attributes.

Desupported Features

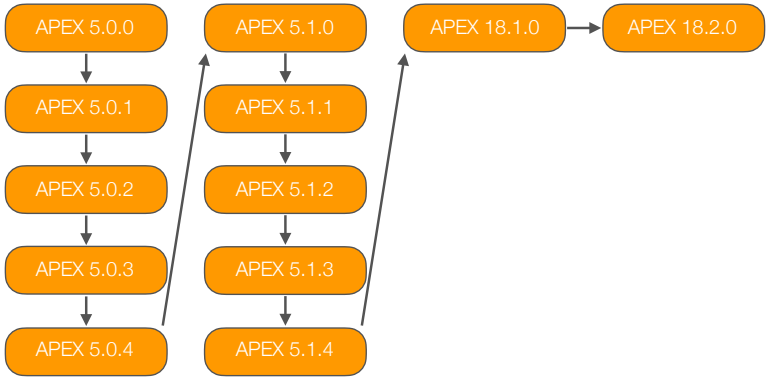
- Desupported features are **no longer available**
  - If a desupported feature has to do with application metadata or APIs, then existing applications may not work as they did previously
  - Oracle recommends modifying the application to replace the feature.

Patching

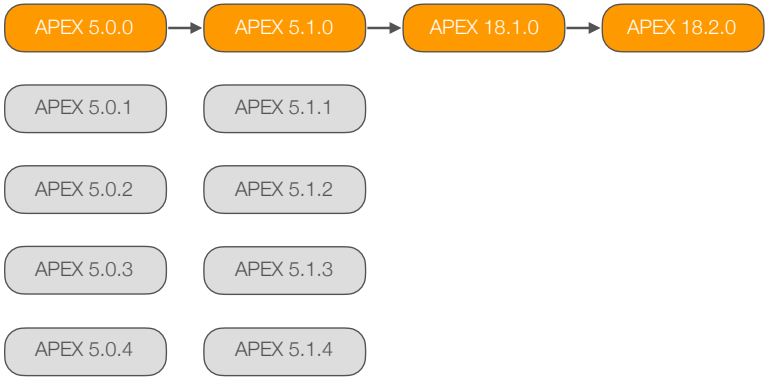
Patching

- **Major Releases** (5.1, 18.1, 18.2) can be upgraded from the APEX builds that are publicly available
  - https://www.oracle.com/technetwork/developer-tools/apex/downloads/index.html
  - All you need is a free Oracle Developer Community account
- **Minor Releases** (5.1.1, 5.1.2, etc) can only be upgraded by downloading the patch from Oracle Support
  - https://support.oracle.com
  - No support = No patches

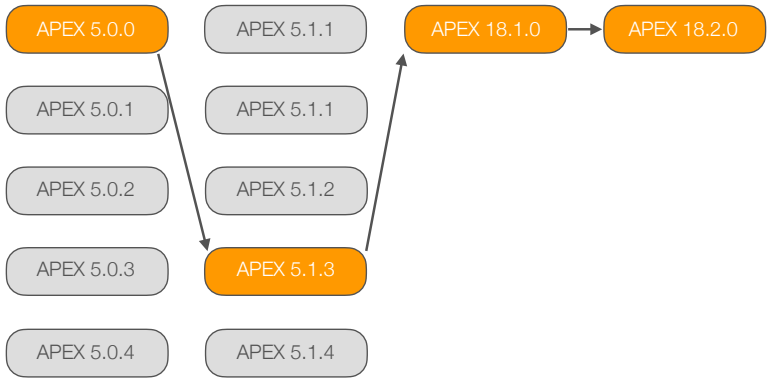
With Support



Without Support



Without Support



Patching

- Be sure to also monitor **CPUs from Oracle**
  - Every once in a while there will be an APEX component
- Goes without saying that **all associated database patches** should also be applied
  - Same reasoning to keep current applies to the database

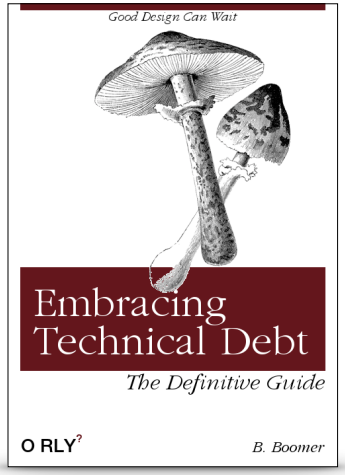
Utilizing APEX core components will result in the most expeditious and hassle-free upgrade & patch cycles

Summary

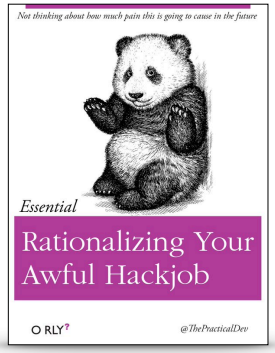
Summary

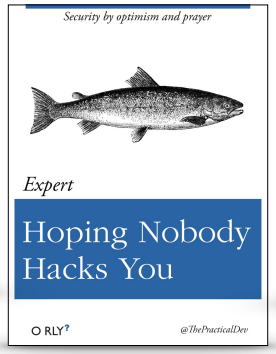
- **Technical Debt** is a real thing
  - You probably have a ton of it today
- APEX is **perfectly capable** of adding to your technical debt
  - Particularly when used incorrectly
- For best results - and less debt - **use APEX how it was intended to be used**
  - Scratch your creative itch elsewhere

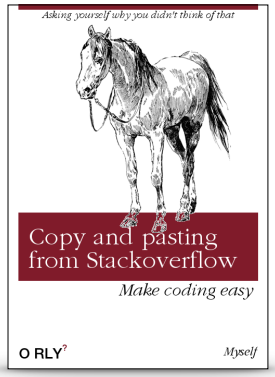
Recommended Reading



Recommended Reading







VISCOSITY
NORTH AMERICA