Managing a Salesforce Project

General advice about Salesforce-related PM

- Pre-Project Phase
- Project Phase
- Post-Project
- Documentation
- Project Reviews
- Auditing a Salesforce Org
- Integration Questions list
- Communities Project Cheat Sheet
- Communities Deployment Hell
- Speaking "Data Maintenance"An intro to Data-related presales

Pre-Project Phase

Roadmap

Define a Roadmap (Integration 1, 2 etc) and keep the different steps digestible in small iterations. Make sure to set clear expectations for these iterations.

Here, it is advisable to focus on Content rather on Timeline.

Separation of concerns

Clearly define who does what.
The Project manager should not be the account manager, similarly the Solution Architect should not act as the Developer.

Define 1 Lead Developer for Code Review - Code refactoring if needed. Also, define a QA person for ensuring quality. QA should be an independent person for spotting errors and disruption testing. Avoid Handover of people.

This applies for Client as well - each person must have a specific function to fill that is documented in the project kickoff. Clear Project Team definition on Clients side. Business Owner, Project Manager. We need 1 person that gives us the orders and makes decisions.

Need a format that's easy to create and consume? Consider using a RACI chart

User Stories

Be precise when writing User Stories by setting limitations/exclusions of the solution.

This means including positive user stories (as a User, I want to be able to view my current Time Entries summary, so I know how many hours I still need to time), as well as negative assumptions (the REST application will not authenticate against Azure directly but go through the client backend, as detailed in User Story UA-15)

This allows to set expectations correctly and helps the customer to amend requests early on in case something is not according to his wishes. Think about what we cannot do rather than solely on what we can.

Data Management

When handling existing organizations or integrations with other systems, it is important to include a Master Data Management Design in your project.

An MDM's main purpose is to define the single source of truth - per table, per field, per integration flow. A MDM allows for accurate and trusted business information of the right quality level, in the right form, at the right time.

Key elements are to define what controls the data at which point, and what happens if the data is not accurately synced.

Project History and Accountability

While deploying bigger projects, it is important to be able to track who deployed what, and in which version each element is.

In order to do so, the recommended approach is to deny any modification by changeset, instead relying on SFDX deployment chains via Git - including admin-only deployments like fields, Flows, or Process Builder. This means that a single person should be identified who will be in charge of scheduling deployments. With that said, have a back-up person so that your main person can take vacation!

Deployments themselves should be scheduled ahead of time, including for bug fixes. They should not number more than one per week, in order to allow for UAT and regression testing.

Assumptions

Are YOUR FRIENDS.

Make assumptions BUT WRITE THEM DOWN AND SHARE THEM WIDELY!!!!. Clearly defined what are the risks if they change the Business owner and what is the consequence. Same for technical assumptions, etc.

Project Phase

Internal Meetings

Meetings that should be held:

- weekly sync-up
- daily standup (30 seconds per person max)
- Pre-sprint and post-sprint meetings
- Account Management meetings in addition (see Sales)
- Client Meetings - Set regular (weekly) meetings with the client to allow for feedback, change requests and sign off of new sprints. Take the opportunity to talk about the status quo of different tasks and how we are performing on planned billed hours. Eg. “ We are doing well on A, on C we are over Budget”

Responsibilities

Make sure the client understands his accountability and the work required on their side eg. minimum 2 days / week available for testing/feedback.

Documentation

Follow Conventions
Prepare a code repository for the client to hand over (for example gitlab etc.)
Prepare technical documentation of Flows/ProcessBuilders/Code
Prepare functional documentation only if paid for by client

Post-Project

Lessons Learned
Make sure to follow the Project Review Process

Make sure we are learning in our process with With Milestones and Project Review activities. What were the learnings on both sides. Weekly iterations for feedback, change request, and for sign off the sprints.

Documentation

Documentation List

The following Documents should be part of most, if not every, project:

- Responsibility Assignment Matrix
- Solution Design
- Integrations Specifications,
- Development Specifications,
- Data Dictionary,
- User Stories spreadsheet,
- Change Request Template,
- Bug-Tracking sheet.

Signing Off

Signing off documentation before any work is done is key to align expectations. Make sure that the person empowered to sign-off is aware of their responsibility!

A list of the different client signoffs is available here:

- at the Project Kickoff:
  - Solution Design,
  - Integrations Specifications,
  - Development Specifications.
- During each PM Meeting:
  - Current status
  - Next Steps
  - Risks & Mitigations
  - Alerts/Other Information
- at the beginning of each Sprint:
  - Sprint Content,
  - User Acceptance Stories,
  - Sprint Timeline.
- at the end of each Sprint:
  - Sprint Outcome,
  - Delta with initial Plan if any,
  - UAT Results
- before Deployments: Please see related wiki article "deployments"

    - Deployment Scope,
    - Deployment Timeline,
    - Risks & Mitigations.
- after Deployments:
    - UAT Outcomes in Production,
    - Validation of GoLive,
    - Which means will be used for Bugtracking

If a “formal” signature is not possible for every step, make sure to have it at least in written form (Email).

Project Reviews

< Linked files on the top left of this article

A Project Review is a process in which you will try to uncover how your project went from start to finish. It is not:

- A focus on elements that did not work,
- A quest to establish blame,
- A simple meeting,
- A highlight of what happened during the project.

The Project Review meeting itself, while necessary, is only part of the entire process.

The result should be clear indications that improve processes and general project outcome for the future.

The main output is an answer to the question “What is going to change in future iterations, and how?”

- What went specifically well, that we can reuse?
- What went normally, but could be improved thanks to what we learned?
- Which actions can we take to avoid the mistakes that did happen?

A Project Review that does not impact future projects is a waste of everyone’s time. Your study should highlight both success and failings that we can use on future projects.

Documents are attached to this article to organize your Project Review.

SFXD Project Review template.pptx

SFXD Project Reviews.docx

Auditing a Salesforce Org

Functional Audit

One workshop mid-audit to discuss their process
One workshop after the audit to discuss findings, step forwards

General stuff to check

Administrator presence - Number of admins, experience, etc. (https://help.salesforce.com/HTViewSolution?id=000007548)
Organization Security - Check the automated health check in Setup > Health Check
User-Friendliness - custom app ? limited number of tabs ? how many fields per page layout ? is LEX-enabled ? Page layouts make somewhat sense ?
Maintainability - using best practices, nomenclatures, not using old notes or old attachments, low-ish number of automations per object, good structure in automations if many, avoid multiple sources of automation on one object.
Usage - have users logged in in the past 30 days ?

Limits

Data limit - Setup > Data Storage
Storage Limit - Setup > File Storage
Object limits - Run Optimizer if possible or check each object limit.
APEX limits - API calls per 24h, errors when tracking a user, etc.

Security

OWD & role reviews
Review of Profiles & Permission sets, flag any VAD or MAD access
Review Permission Set Groups & Muting Permission Sets
Review of any external access
Review Sharing Rules

Data Model

Review of limits and object usage
Review of field usage
Review of general architecture
Solution design of Refactoring if needed

Data

Duplicates ?
Reportable ?
Owners make sense ?
Old records ?
Quality ?

Automation

Number of Processes/Workflows per object
Process Builder best practice audit
Flow best practice audit

- How many flows are there ?
> few: not used, flag it as admin probably doesn't know about them, check how the flow looks internally to check
> medium: can be anything, but OK
> A lot: are they well named ? If yes yay, that's generally an admin that knows their shit. If not I'm in for a world of hurt straight out the bat.

- how big are the flows ?
> small : don't care
> medium: don't care
> big: see if there's squares in the same order a few times, that means they should be using subflows. you'll notice it
> huge: should be using subflows anyway

- variables
> no input variables : they're using it as a script, train admin, but at least it's easy shit
> only input variables : they have no idea wtf input vars do, train them and check the flow for other bullcrap
> normal usage: yay

- elements
> are the elements named consistently ?
>> Yes: yay
>> No: fuck

> are all elements present used ?
>> yes: YAY:
>> no: fuck

> are all elements that could be reused reused, or are there a ton of throwaway elements ?
>> tons of shit that gets used once because they don't know how to use assignments and vars: fuck
>> reusable stuff: yay

- structure
> are the DMLs outside of the flow main structure ?
>> yes: YAY
>> no: fuck

> Are there elements in loops other than assignments or decisions ?
>> yes: fuck
>> no: YAY

- maintainability
> can the flows be read ?
>> yes: phew
>> no: fuck

> do you know wtf they're here for in less than 30 minuets ?
>>yes: somebody did their job right
>>no: fuck

> are the structures of the flow consistent and make it easy t maintain (subjective)?
>> yay
>> fuck

> are there flows that should be APEX or other shit ?
>> yes: list them, explain why they shouldn't be that way
>> no: cool beans

Validation Rules best practice review
Workflows review

APEX

High-level APEX review: naming, basic structure
Code audit: standard best practices, optimization
Architecture audit
Solution design of refactoring if needed

Generation of the Audit report

Integration Questions list

- Which direction is the sync going from ?
 - SF > other system ?
  - is that system accessible via the internet ?
  - does it support OAuth2 ?
  - does it support TLS1.2 ?
  - Does it have a REST or SOAP API ?
 - other system > SF?
  - can it leverage standard SF APIs ?
  - does it support OAuth2 ?
  - does it support TLS1.2 ?
 - both ?
  - all of the above, plus:
   - do the synced tables overlap ?
   - if yes, which system is the master ?
   - how should conflicts be resolved ?
- When is data synced ?
 - on record change ?
 - on user action ?
 - at set time ?
 - a combination of the above ?
- What is the amount of data to be synced, in number of records, per table ?
 - If > 1mil yearly:
  - should we archive old records ?
   - where ?
  - should we delete old records ?
  - are BigObjects a possibility ?
  - can we look into having an external data warehouse where we have the data and SF only pulls what is needed ?
- is there a 1/1 correlation between tables in SF and backend, or should we do a custom mapping between tables ?
- are there limits in he other system we need to be mindful of ?

Pre-scoping questionnaire

Does the client backend have any access restrictions?

- **Specific IPs to whitelist?**
- **Proxy/limited access/etc?**
- **Does the backend support TLS1.2 minimum?**
- **Does the backend support OAuth2?**
- **Are there limits to the number of API calls that can be made?**
- **Are there limits in payload size?**
- **Are there limits to the delay in answering a specific call?**
- **What is the average response time of the API?**

Is REST the only viable integration solution?

- **Do all the data flows have to be real time?**
- **Can some data flows be handled via batch nightly loads instead of REST integration?**
- **Is the tight coupling REST integration forces a problem for the future?**
- **In case of real-time data sync, has events-based architecture been considered?**

Does the client have documentation they can send over?

- **Swagger or equivalent?**
- **Lacking that, basic pdf documentation?**
- **Lacking that, examples of calls and responses?**
- **Lacking that, can the client provide a list of supported methods (GET, PUT, POST, etc), and expected formats (JSON, XML) ?**

Are all data flow operations possible in the current backend?

- **Do endpoints exist for all specified data operations?**
- **If yes, are they considered stable?**
- **If no, can new endpoints be developed, or existing endpoints adapted?**
- **If yes, in which timeframe?**
- **If yes, is this viable for the current project?**
- **If multiple calls depend on each other**

Communities Project Cheat Sheet

COMMUNITIES IN SF CHEAT SHEET

Shit to check
- what do my users need access to ?
- opps ? > Partner
- reports ? > Probably Partner
- which records do my users need access to ?
- stuff they own or are explicitely linked to ?
- or shit that'll need sharing rules ? > Partner
- How many users and how often do they login ?
> Changes license type, either user based r login based
- How much Files will they store ?
CommunityUsers don't bring loadsa file storage watch out
- How much Data will they store ?
Same, but data in SF isn't super expensive so meh
- Branding ?
Hard to do
- Needs LWC ?
Maybe hard to do depending on class access
- SSO ?
Not hard but always a bother
- Login Pages
Known bug where activating communities may not provision Login elements. If Login page full empty, contact SF support.

Stuff that's hard
- Sharing
Either you're full customer community and there's only Sharing Sets
Or you're in Partner world and you get the worst of every world.
Community Users owning record sis general meh
Sharing records with them based on sharing rules sounds great but is sometimes hard due to the Partner Role structure which just merges with the main one like a big Tick
it's just annoying
Make sure to have a valid sharing diagram, really
- Deploying
Deploying communities is ALWAYS shit

Tools will tell you they ease the process. They lie. There' sonly so many ways to deploy - change sets, API, SFDX. All of them have issues with communities. Maybe you'll get lucky, maybe you won't, who knows.
There's arcane bugs like if your objects have the same beginning of a name Pages won't be carried over because the API deployment truncates the namings
just deploying is HARD.

In general consider it easier to just rebuild the community from scratch in prod rather than deploy. Because that's what you may end up doing.
- Branding
We're not UI specialists.
Either they go standard, or they hire an Agency. Anything in between they can go fuck off.
- ORDERS
ORDERS SUCK
THEY DO
DON'T USE ORDERS IN COMMUNITIES
PERIOD
- List Views
remember to change sharing for all list views if any exist before
Also remember to make the list views available for your comm users if they need it

Stuff that's easy
- building pages is nice
- base branding is nice if no custom
- Flows are :100:
- No "separate database", just different view into same system makes shit nice

Considerations
LOCK DOWN EVERYTHING and then only open up what's necessary
GDPR real people, srsly don't open up shit that doesn't need to be.
Licensing is the main way to fuck up your Comms project
Data access is the second way
Look and feel is the third
Anything else is easy

Awesome new shit
- Record based Audiences mean dynamic pages for users based on criteria
- Flows in comms are so useful they might as well replace the entire thing with that

Weird shit that doesn't matter much
- CMS is a weird offering that kinda only half exists, almost no one at SF has seen it
- Translations can be wonky and still rely on the old Site.Com Builder in the backend but you shouldn't have to go there to edit translations now
- there' s3 different places to look at comm shit:
- the comm builder
- the Site page which is where you see the Guest user access

- the Site Builder page which you can access through Workspace > admin > a small link hidden within one of the pages, that only handles very few things and you can ignore

Communities Deployment Hell

so
Communities
they are split into multiple things in the backend
a Network
a Site
and some other free floating shit like the guest user, profile access, etc.
Network is what you think about when you read Communities
it's where the comm builder etc lives
the Site is how they exposed it to the world. it's a legacy feature wrapped in a new one
the Site as detailed in my longer rant used to be where you handled translations and shit, because OF COURSE that was a good idea, anyway.
when deploying a community you have a two options
"bundle" it with a Lightning Bundle as they call it, which is basically a template of your community
this works nicely but doesn't include everything about the comm (I forget what exactly but some stuff like branding etc wouldn't get carried over as well as some custom pages)
or pushing the Network and the Site via Changeset or API.
now via API is what Gearset does. The good point of API deployment is that you can target the network site, and related StaticResources and other shit directly
it's easy enough
the bad thing is that API deployments rely on the backend structure being sane, which is isn't.
Example: if you have two object `REALLYLONGPREFIX_stuff__c ` and `REALLYLONGPREFIX_shit_c `
the backend stores a truncated version of the page
so if you do custom pages
this results in you having the pages being possibly misattributed to the other object
or vice versa
or overwriting the other page
at random
everytime
on deploy
Q U A L I T Y.
the API deployments are also less fault tolerant than changesets meaning if something's missing it'll just completely fail and some of the community errors are quite arcane
specifically the ones related to Site deployment
so, changesets ?
well changesets aren't much better
you can still get the arcane Site deployment errors, thoughtless of them because it resolves some missing shit on its own for some reason ????

but what's nice is that the community elements will save in whichever order
so for example
you can completely have a community that'll deploy
but fail because the Asset for the custom logo of the banner is missing
even though you included it in the changeset
so now you gotta deploy the asset alone
and then deploy the community
which will STILL say that the deployment FAILED but show you a green checkbox saying "deployed successfully"
which will actually mean it deployed successfully because the only failed elements are the Assets.
nope, still not kidding.
now for easy ass communities you might get a deploy that goes well
and SF may hotfix some of those issues
but in the meantime it's better to consider the entire deployment process hell
plan more time for it than needed
and if it's simple in the end well that's awesome.

Speaking "Data Maintenance"An intro to Data-related presales

Glossary of Partners

Mulesoft

Salesforce-owned company.

Salesforce will tell you about Mulesoft, and forget to tell you if they’re speaking about Anypoint, Composer, or anything else about Mulesoft. Make sure they are targeting the discussion in ways that serve the client see data volumes, mappings, complexity) rather than the product.

Talend

The Free version is limited and generally is just used to cross load CSV files. The paid version is very powerful but requires an IT team to wield properly, and has setup costs for us regarding how to set the environment in place.

Jitterbit

Is bad. Run away. It used to be the king, but lack of updates, bad infrastructure and bad support lead to it losing ground over the last years.

Boomi

A paid ETL by Dell. Powerful, used by US corporations, but paid. Rarely seen in the wild unless the client already has a license.

Informatica

A paid ETL by Informatica. Powerful, used by US corporations, but paid. Extremely rarely seen in the wild unless the client already has a license.

Kafka

An event bus by Apache. Used by Event Driven Systems

Glossary of Technologies

API

An Application Programming Interface (API) is a set of functions, procedures, methods or classes used by computer programs to request services from the operating system, software libraries or any other service providers running on the computer. A computer programmer uses the API to make application programs.

MDM

Master data management[1] (MDM) is a technology-enabled discipline in which business and information technology work together to ensure the uniformity, accuracy, stewardship, semantic consistency and accountability of the enterprise's official shared master data assets.[2][3]

In simpler terms, it is the act of defining which system has the correct data, where, when, and how it is kept up to date.

Clients often request an “MDM”, which actually just means a “centralized system of record”, meaning a database where they know the data is correct and should always prime in case of data differences with other systems

ETL (Extract-Transform-Load)

A software tool that extracts data from a source system, transforms the data (using rules, lookup tables, and other functionality) to convert it to the desired state, and then loads (writes) the data to a target database.

Web service

A Web service is defined as "a software system designed to support interoperable machine-to-machine interaction over a network". Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

REST

Representational state transfer (REST) is a software architectural that was made to guide the development of the World Wide Web. Systems which implement REST are called 'RESTful' systems. REST documents a way for computer systems to communicate with each other using HTTP requests.
It is supported by most recent players, is flexible and cheap.

It is also less secure than SOAP by design, and for high volumes, Events can be better suited.

SOAP

SOAP is a protocol used in computing. Web services use this protocol to communicate. SOAP uses XML to encode a message. It uses other application-layer protocols, for transport, and content negotiation, for example HTTP and Remote procedure call.

It is less flexible than REST and harder to implement, but it offers more security and some calls are specific to SOAP.

GraphQL

An API type that’s similar to REST but has technical differences in implementation and scope of data recovery. Great if multiple calls need to be done of varying scopes on the same endpoint.

Web socket

Much like REST, it is an HTTP API protocol. It has way less flexibility but is great if you want to “just push a message somewhere”, if that message corresponds to a very specific format.

Events

Events operate on the opposite of REST/SOAP calls. In REST/SOAP you tell a system what you want it to do, and add information needed for the action. Events just say “something happened, here’s the data about that”. It becomes the receiving system’s job to interpret the action to do.

Events are asynchronous, and by nature harder to manipulate and ensure than REST/SOAP calls. It’s great for high-volume, low-latency situations, but expensive.

ESB (Enterprise Service Bus)

REST and SOAP historically integrate two different platforms directly. These platforms become “coupled” - if one changes, the other must change to allow the integration to continue.

An Enterprise Service Bus is a platform that sits in the middle of these integrations. All platforms speak to the ESB, and the ESB then manipulates data, streams, events, and whatever else is necessary to allow the platforms to get the information they need back.

Setting up an ESB is costly, and generally leads to restructures in existing integrations so they leverage the new ESB. It does however lower the cost of future integrations, and lowers platform coupling.

It is a good idea to implement an ESB when you have at least 4 platforms speaking together, and it can be valuable to look at it for lower numbers.

Batch

The default Data Loading mode for Data Loader and REST calls.
Accepts Data passed via REST, in batches. Processes these batches *synchronously* and then

returns the results as a response with the same number of records as in the original batch, with a status code.

The default batch size in Data Loader is 200. The number of batches submitted for a data manipulation operation (insert, update, delete, etc) depends on the number of records and batch size selected.
One API call is used per batch, which can lead to limit issues for big loads.

Bulk

A different Data Loading mode, usable via Data Loader or REST calls.

Accepts Data passed as a CSV file which must be sent to the server in a series of REST calls. Once all the data has been received, a final call tells the bulk to start. It then processes these batches *asynchronously* and returns the results to the batch, which must then be downloaded via REST calls.

The default BULK size in Data Loader is 2000. The amount of records loadable is by nature very high (a few million), and as such this API is recommended for big data transfers.

Event-Driven Architecture

A situation where the client already uses Event-based systems and expects you to implement a receiving Event Bus and get Events for integrations. See Events.

Database

Often conflated with Relational Database Management System, actually just means a place where data is stored. Can be relational, graph based, events based, whatever. If “database” is said, try to see which kind.

Data Warehouse

Often conflated for “lots of tables”. Actually means place where data frim multiple systems are stored. Doesn’t have to mean that the data is transformed to serve an MDM - you can just store multiple systems and call it a day.

Data Lake

Often conflated for “lots of tables”. Actually has nothing to do with tables, and defines an architecture for data storage, with a heavy focus on data “flatness”, hence “lake”. The data can be structured, semi-structured, unstructured - meaning a Data Analysis team will be needed to use it properly.

If the client is misusing this term, it’s fine. If they’re using it correctly, the complexity of your project just went up.

Data Archival

Taking data from a system and storing it in another when it’s no longer useful but you don’t want to lose it. Generally done for Cost considerations - storing in a local postgredb is cheap.

Glossary of Volumes

Data Storage

Amount of records salesforce stores. Records in Salesforce are generally (exceptions apply to tasks, events, email messages) abstracted to 2kb per record. Storage is expensive in Salesforce, keeping it reasonable generally lowers project cost.

File Storage

Amount of ContentDocument Salesforce stores. Very expensive, and Salesforce does document management poorly. You might want to look into third party solutions.

LDV (Large Data Volumes)

Above 500000 rows in a single table, LDV applies. This is a key word for Architects that will understand they need to watch out for volumes, flows, api calls, storage over time etc.