What Worries Salesforce Users Most When Using Their Orgs

salesforce

46%

HITTING GOVERNOR LIMITS

31%

BUGS CAUSED BY CUSTOMIZATION

23%

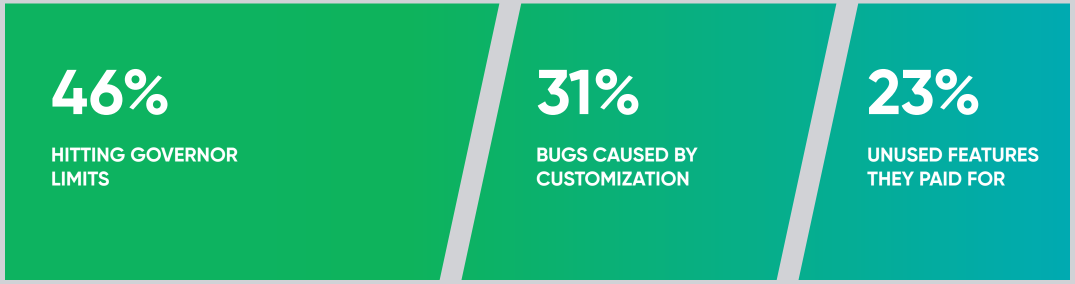
UNUSED FEATURES THEY PAID FOR

brimit

Salesforce is a robust platform for digitizing business operations and automating processes in an organization. However, issues sometimes arise when running the platform. I asked current Salesforce users about what worries them most when using the platform. In this blog post, I would like to present the most common answers to this question and discuss some approaches to dealing with known issues in Salesforce.

WHAT BOTHERS SALESFORCE USERS THE MOST WHEN USING THEIR ORGS

Results of the poll by Slava Pautaran



Hitting Limits

46% of my respondents sometimes encounter issues with hitting limits. Let's see what limits can be hit and how to avoid this.

Apex governor limits

To prevent the monopolization of Salesforce's shared resources in a multi-tenant environment, the concept of governor limits is applied. Apex code won't work when you exceed these limits, which means you should try to review it. There are two big problems here.

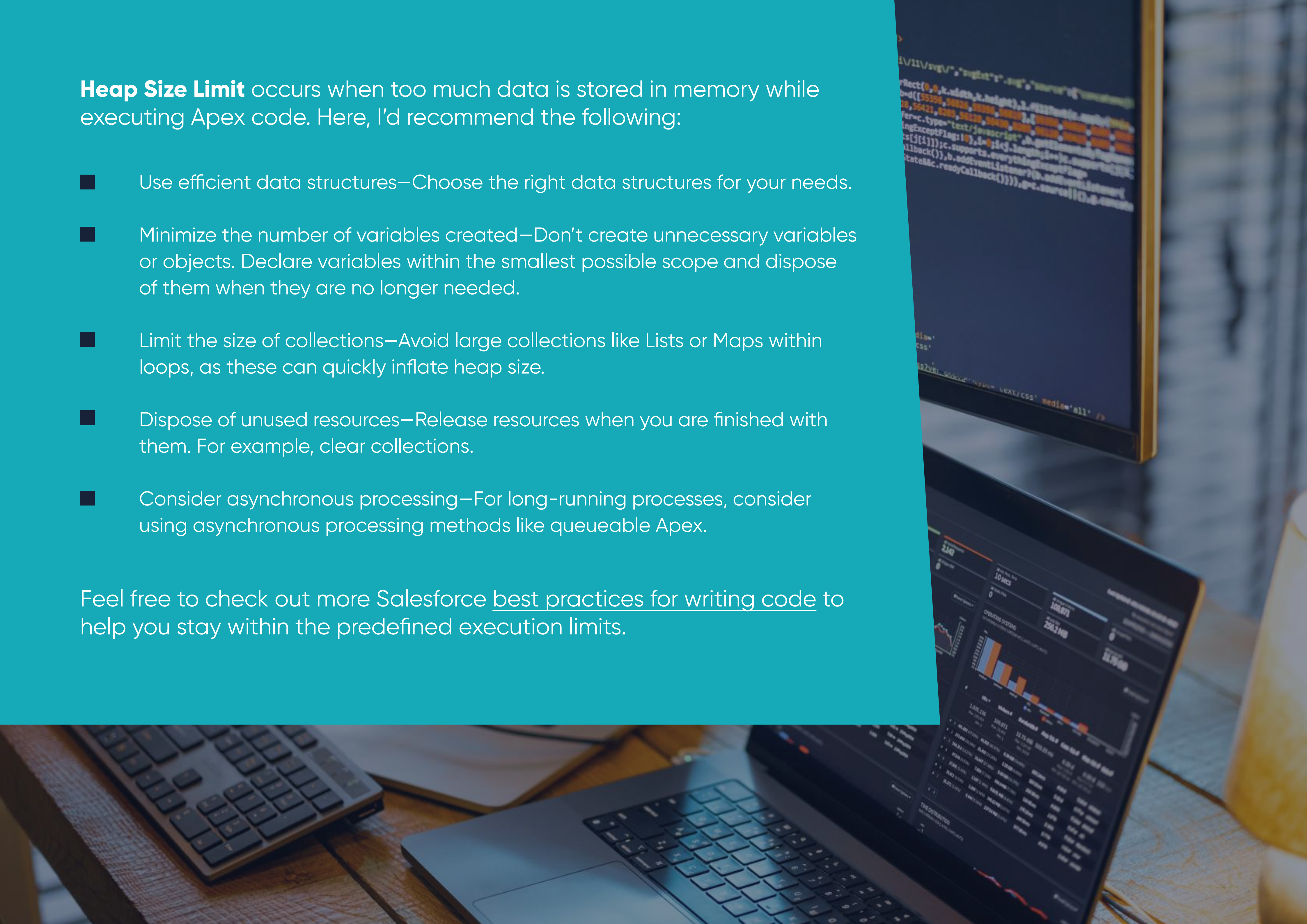
CPU Time Limit Exceeded arises when automations (Apex, Flow, etc.) take too long. There are some best practices for such situations:

- Avoid complex calculations—Complex logic can be CPU-intensive. Try to simplify your code and use clear functions. If you require resource-intensive calculations, consider using Salesforce Functions, or build a Heroku app for this.
- Review how loops are used—Loops, and especially nested loops, consume a significant amount of CPU time. Consider using collections or bulk operations to eliminate the need to cycle through records.
- Avoid recursion—Recursion can cause you to quickly reach the CPU time limit. Be sure to avoid endless recursion.
- Asynchronous processing—The queueable/future and batch Apex methods allow you to execute long-running or resource-intensive processes in the background without impacting the current transaction.
- Bulkify code—Wherever possible, write code that can handle large amounts of data efficiently. Use bulk patterns to process data and avoid writing code that processes one record at a time.

Heap Size Limit occurs when too much data is stored in memory while executing Apex code. Here, I’d recommend the following:

- Use efficient data structures—Choose the right data structures for your needs.
- Minimize the number of variables created—Don’t create unnecessary variables or objects. Declare variables within the smallest possible scope and dispose of them when they are no longer needed.
- Limit the size of collections—Avoid large collections like Lists or Maps within loops, as these can quickly inflate heap size.
- Dispose of unused resources—Release resources when you are finished with them. For example, clear collections.
- Consider asynchronous processing—For long-running processes, consider using asynchronous processing methods like queueable Apex.

Feel free to check out more Salesforce best practices for writing code to help you stay within the predefined execution limits.



API request limits

Salesforce also restricts the number of incoming API requests per organization/24 hours; the total number available is calculated by multiplying the number of active licenses by the quota per license and adding the number of base licenses available.

So, if you can't gain access to data in your org using a third-party app integration, it would be best to use a different API that saves your request or use bulk APIs.

For example, you might be using multiple subsequent requests to create an account, contact, or opportunity from a third-party application such as a public website. Here, you are using three API calls, which is a lot. Instead, you could use Composite or sObject Tree REST resources to optimize your API calls. In this case, the entire request counts only as a single call toward your API limit.

Also, your application might be creating hundreds or thousands of data entries hourly or daily, which means that it is implemented as real-time or near-real-time integration. In this case, you can question whether it's required to have all of these integrations in real time or near-real time. You can save a lot of API calls if you make data replication—which runs every 15 min, hourly, or even daily—a scheduled job instead. This way, you can use Bulk API to keep your API call quota from hitting the limit.

Salesforce recommends certain best practices for using certain APIs. Follow the link to find your use case.

Storage limits

Salesforce encourages users to scale up as an organization grows and the volume of business operations expands. As the volume of data to be stored increases, so does the time it takes to perform certain operations. Which increases the time required to perform certain operations. Salesforce storage is divided into two categories—file storage and data storage. Both of them need to be optimized.

To optimize work with files, organize them properly in folders, practice file collaboration and version control (instead of duplicating files), and review old files and files owned by deactivated users. There are also some best practices for working with large data volumes (LDVs). These include archiving unnecessary data, using mashups and selective queries, and adopting more efficient ways to load LDVs into Salesforce. You can also check out the best practices for deployments with LDVs.

Bugs caused by customization

Any custom-developed software usually has bugs, as it is not possible to foresee everything and plan for every edge case and dependency. However, there are some practices that you can follow to minimize the number of bugs and their impact:

- Use Git for source control, branching, and to create a release strategy
- Use source-driven development with SFDX
- Implement coding guidelines and development best practices, including unit testing best practices for both Apex and LWC code
- Implement an efficient environment strategy for Salesforce development and operations
- Create scratch orgs for development and sandboxes for dev integration, QA, UAT, and hotfixes
- Implement peer code review and static code analysis
- Implement automated regression testing
- Train QAs to understand the specifics of Salesforce (Passing admin certs is usually a huge help.)
- Make frequent, more granular releases (CI/CD)
- Conduct smoke and sanity testing for every release

Underused subscription features

Sometimes, Salesforce users purchase licenses for products that end up never being used. They either don't need the products or don't know how to start using them or take advantage of certain features.

First, check the summary data on the company information page in the setup menu to start dealing with unused licenses and underused product features. Then, you'll probably need a competent Salesforce administrator to help you optimize your tool set.

For example, compare your implementation with the following useful features:

- Account, Opportunity, Case Teams
- Lead, Opportunity Scoring
- Prediction Builder & Next Best Action
- Einstein Bots
- Knowledge Articles Recommendations
- Email Integration
- Omni-Channel Routing
- Chatter
- Field History Tracking
- Forecasting
- Dynamic Forms
- Permission Set Groups
- Restriction Rules
- In-App Guidance
- Pipeline Inspection
- Scoping Rules
- DevOps Center
- DevHub & Scratch Orgs

  Flow Orchestrator

Many of the features above are available OOTB. Feel free to look for them and try to apply them yourself.

Request a FREE Salesforce health check from a Salesforce Certified Technical Architect

https://www.brimit.com/technologies/salesforce-health-check

Brimit is a team of Salesforce experts

https://www.brimit.com/technologies/salesforce