Frameworks
eine Herausforderung für Software-Architekten

Die Verwendung von Frameworks erfolgt aus zwei verschiedenen Richtungen: Entweder wird am Anfang eines Projekts entschieden die Entwicklung auf ein Framework aufzubauen, oder während der (Weiter-) Entwicklung werden wiederkehrende Funktionen ausgelagert, so dass sie zentral für andere (neue) Bausteine zur Verfügung stehen. Beide Herangehensweisen verlangen Fingerspitzengefühl und eine ernsthafte Abwägung, ob zum Beispiel ein eigenes Framework eingesetzt oder (neu) entwickelt wird, oder ob ein fremdes Framework übernommen wird.

Für ein fremdes Framework spricht oft, dass man sich die Umsetzung von komplexen Funktionen wünscht, so dass man nicht alles zu Fuß erledigen muss. Das setzt ein gewisses Vertrauen auf den fremden Code voraus. Da bekanntlich kein Programm ohne Fehler ist, ist fremder Code gleichzusetzen mit einer eigenen Überprüfung oder mit einer Option auf ständige Aktualisierung durch den Hersteller, meist eine Community. Im letzteren Fall werden Fehler dort schnell bekannt und das Framework kann schnell zu einem Sicherheitsrisiko werden, wenn man nicht oft genug patched. Die Überprüfung des fremden Codes mag an und ab zwar ganz lehrreich sein, aber gerade dieser Arbeit hat man ja versucht aus dem Weg zu gehen.

Entwickelt man über Jahre hinweg ein eigenes Framework, bedeutet dies, dass man dieses ausschließlich selbst aktualisieren muss. Das kann eine Menge Arbeit bedeuten, zumal die Erkenntnis für eine Aktualisierung oft mit neuen Projekten einhergeht. Aber schließlich dann hat man kaum Zeit für die Erweiterung oder das Refactoring eines alten sicher noch irgendwo im Einsatz befindlichen Frameworks. Die Abwärtskompatibilität hat man zwar selbst in der Hand, aber vielleicht nicht die erforderliche Erfahrung, diese von Anfang an in der Framework-Architektur zu berücksichtigen. Bastelt man in Zeiträumen zwischen Projekten, sofern es diese gibt, an der Weiterentwicklung des eigenen Frameworks, so besteht die Gefahr, dass Funktionen eingebaut werden, die nie angewendet werden, oder die im letzten Projekt notwendig gewesen wären. Dieses kann oder darf man nicht mehr anfassen, wie will man diesen Mehraufwand auch dem Kunden erklären oder verkaufen, oder der Einbau geschieht als Testlauf, welcher dann aber nicht garantiert, dass die Anwendung des Frameworks im nächsten Projekt ähnlich wird.

Ist aber viel Energie in die Entwicklung eigener Bibliotheken oder in die Anwendung fremder Frameworks geflossen, möchte man diese Erkenntnisse Gewinn-maximieren, indem man sie wieder anwendet, auch wenn im neuen Anwendungsfall eine andere Lösung sinnvoller gewesen wäre.

Wie kann man diesem Dilemma entgegenwirken? Es gibt ein paar einfache Regeln, die kritische Fragen für und wieder (alter) Frameworks fördern:

1. Die strikte Verfolgung einer drop-down-Entwicklung verhindert lange Investitionen in Framework-Funktionen, die später nie angewendet werden.
2. Kurze Iterationen mit frühen sichtbaren Ergebnissen für die Zielgruppe lassen wenig Zeit und Kapazitäten für Bibliotheks-Kapriolen.
3. Beteiligt man sich selbst an der Entwicklung von Community Frameworks, bleibt man am Puls der Zeit und verpasst keine (Fehl-)Entwicklung. Aufpassen muss man hier nur, dass man sich nicht verzettelt und mehr für die Community tut, als diese für einen selbst.
4. Zumindest der Versuch, zu verstehen, wie ein Framework seine Funktionen umsetzt, bewahrt einen vor Überraschungen, und verschafft einem manchmal die Erkenntnis, das hätten wir auch selbst zu Fuß (vielleicht besser) geschafft.
5. Pausen bei der Adaption eigener Frameworks darf man sich nicht gönnen, sonst verpasst man vielleicht den technologischen Anschluss an Sicherheitsstandards oder Fortentwicklungen in der Programmiersprache.

6. Planen Sie von Anfang an, wie Sie verschiedene Versionen desselben Frameworks einsetzen können, zum Beispiel mit Namespaces. So können neue Komponenten auf neue Funktionen einer neuen Version aufsetzen.
7. Bei eigenen Frameworks sind die Abhängigkeiten der einzelnen Module des Frameworks so gering wie möglich zu halten.
8. Gibt es innerhalb des Frameworks unterschiedliche Abstraktionsebenen, dann sollten diese klar voneinander unterscheiden werden und die Schnittstellen zueinander einfach gehalten werden.

Schließlich schadet es nicht, wenn individuelle Lösungen entwickelt werden, weil diese oft einfacher sind, direkt auf Programmiertechniken aufsetzen, besser optimiert werden können und das Wissen im eigenen Umfeld konzentriert werden kann. Es muss nicht immer ein Framework sein, manchmal reicht es auch schon, die Möglichkeiten der Programmiersprache auszureizen, oder gesunde individuelle leicht zu wartende Architekturen zu entwerfen. Das Handwerkszeug dazu kann man sich ja ein wenig bei großen Frameworks abschauen. Ein gesundes Maß an Selbstvertrauen ist der erste Schritt zur Unabhängigkeit.