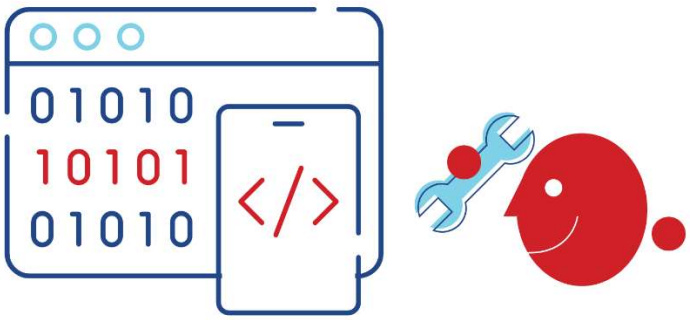
ALTKOM SOFTWARE & CONSULTING

Raport stanu technicznego SystemuX pod kątem jakości kodu.



Spis treści

Kluczowe wnioski na podstawie przeprowadzonych badań:

1. Stan techniczny projektu jest bardzo dobry. W skali 1-10 osiągnął on wynik 9.09, gdzie dla projektów o tej wielkości i czasie życia, wynik większy lub równy 5 jest wynikiem dobrym.
2. Jakość i złożoność projektu jest stabilna i od dłuższego czasu nie ulega pogorszeniu. Oznacza to, że na chwilę obecną nie ma konieczności prowadzenia dodatkowych prac nad zmniejszeniem długu technologicznego. Obecnie stosowane podejście zapewnia wysoką jakość kodu. Zaleca się jednak monitorowanie jakości w modułach policy, tarrification i web, ze względy na ich gorszy stan od reszty systemu oraz ich kluczową rolę.
3. Postępujące od dłuższego czasu zmniejszanie zespołu, skutkuje tym, że ponad 70% kodu została stworzona przez osoby już niepracujące w projekcie. Stanowi to duże zagrożenie w przypadku konieczności wprowadzenie zmian w części kluczowych modułów: tarrification, policy, product. Moduły te są jednak stabilne i w ostatnim czasie nie były poddawane dużym modyfikacjom. W przypadku planowania prac w tych obszarach zaleca się uwzględnienie czasu na poznanie kodu przez obecny zespół.
4. Znaczny rozmiar projektu oraz duża utrata wiedzy oznaczają wysokie koszty on/off boardingu. Należy postarać się o minimalizację rotacji w zespole projektowym. Z osób pracujących nad projektem, obecnie szczególnie istotną rolę odgrywa *ProgramistaB*. Jego odejście z zespołu spowoduje utratę wiedzy w modułach claim i billing. W połączeniu z niskim poziomem opanowania wiedzy przez zespół w innych kluczowych modułach, stanowi to duże zagrożenie dla dalszego rozwoju projektu. Wprowadzając nowe osoby do zespołu należy zwrócić uwagę na pozyskanie wiedzy w kluczowych modułach: polisy, billing, taryfikacja, web.
5. Średnia wydajność programistów utrzymuje się od dłuższego czasu na zbliżonym poziomie. Oznacza to, że rozwój systemu przebiega prawidłowo oraz że nakłady na utrzymanie odpowiedniej jakości przynoszą efekty. Ważnym czynnikiem jest też duże doświadczenie zespołu - większość programistów pracuje w nim ponad rok.
6. Zaleca się ujednolicenie podejścia do zarządzania zadaniami w JIRA. Duża ilość statusów, długie czasy oczekiwania na code review czy długie czasy między przekazaniem do testów a zamknięciem uniemożliwiają wykonanie automatycznej analizy kosztów. Alternatywą jest rejestrowanie czasu pracy spędzonego na danym zadaniu manualnie bądź automatycznie przy pomocy odpowiednich pluginów JIRA.
7. Wykorzystywane w projekcie technologie są w większości w nowych wersjach oraz są na bieżąco uaktualniane. Wyjątek stanowi tutaj Liferay Portal CE 6.2 oraz Angular 1.x. W przypadku Angular 1.x oficjalne wsparcie przez twórców przewidziane jest do grudnia 2021. W przypadku Liferay wsparcie takie już nie występuje. Zaleca się opracowanie planu migracji komponentu Liferay jako elementu działającego w publicznym internecie i mogącego stanowić poważne zagrożenie bezpieczeństwa.

Wstęp

Przedmiotem analizy jest *SystemX*. Analiza została wykonana na bazie całości kodu z repozytorium *<link-do-repozytorium>* według stanu na dzień 5.06.2020 r. Analizowano kod od 2015-09-02 r.

Analizowane były również dane z systemu zarządzania projektem JIRA <link-do-jira> z projektów JiraProjektA i JiraProjektB.

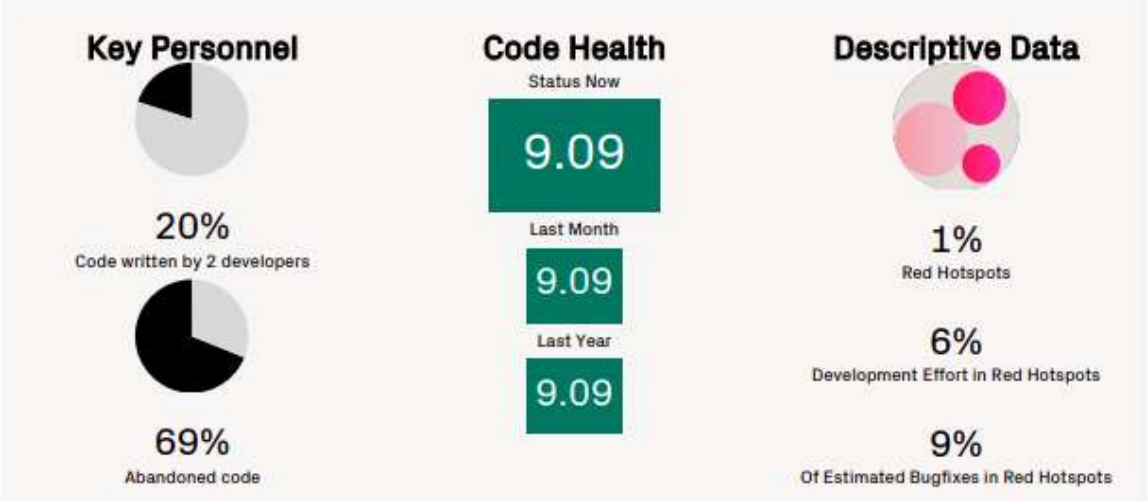
SystemX jest aplikacją back office służącą do kompleksowej obsługi zakładu ubezpieczeń. Ma architekturę opartą o mikro-usługi. Kod napisany jest głównie w językach Java i JavaScript.

Podejście do analizy

Analiza została wykonana z wykorzystaniem narzędzia **CodeScene.** Jest to narzędzie, które wykorzystuje tzw. behawioralną analizę kodu (behavioral code analysis). Główną różnicą między behawioralną analizą kodu, a tradycyjnymi narzędziami takimi jak SonarQube jest to, że analiza statyczna działa na konkretnym "snapshocie" kodu i to on podlega analizie. CodeScene bierze pod uwagę historię zmian i ewolucję całego systemu w czasie. Dzięki takiemu podejściu poniższy raport posłużyć może do priorytetyzacji prac nad długiem technologicznym i poprawą jakości kodu.

Dodatkowo analiza behawioralna daje też cenne informacje, które są niewidoczne w samym kodzie źródłowym, takie jak miara niezależności/autonomiczności zespołów i ryzyka z nią związane czy też ryzyka związane z off-boardingiem poszczególnych członków zespołu.

Stan techniczny projektu



Analiza wykazała, że projekt jest w bardzo dobrym stanie technicznym. Poziom stanu zdrowia kodu to 9.09 (w skali 1-10), co dla projektu trwającego ponad 5 lat, z tak dużą ilością kodu (około 1 mln linii) jest wynikiem bardzo dobrym. **Dla projektów w takiej fazie rozwoju za dobre przyjmuje się wartości większe od 5**. Stan kodu jest stabilny i nie uległ pogorszeniu w bieżącym roku.

Projekt liczy obecnie **877,225** linii kodu. Aktywnie pracuje w projekcie **7 programistów**.

20% kodu zostało napisane przez 2 programistów, co sugeruje, że w projekt zaangażowany był duży zespół i wiedza nie jest skupiona w rękach 1-2 osób.

Hotspoty, czyli pliki z kodem, które są najczęściej modyfikowane stanowią zaledwie 1% całego kodu. Oznacza to, że kod jest stabilny, napisany zgodnie z dobrymi praktykami programistycznymi (zasada otwarte/zamknięte, ang. Open/closed principle).

69% kodu oznaczone jest jako "kod porzucony". Oznacza to, że 69% kodu zostało napisane przez programistów, którzy nie uczestniczą w projekcie aktywnie od przynajmniej sześciu miesięcy. Niesie to za sobą duże zagrożenie, ponieważ kod przez nich napisany dotyczy kluczowych obszarów projektu takich jak zarządzanie polisami, taryfikacja czy zarządzanie produktami.

System

Analiza Hotspot

Poziom modułów

Hotspot to takie fragmenty kodu projektu, które są najczęściej modyfikowane w trakcie życia projektu. **Analiza statystyczna wielu projektów pokazuje, że we wszystkich rozkład zmian jest taki sam - największa częstotliwość zmian (czyli większość prac deweloperskich) występuje w bardzo małej części kodu (poniżej 15%). Działania optymalizacyjne i naprawcze należy więc koncentrować w tych obszarach.**

W przypadku *SystemuX* analiza wykazała, że wykluczając kod o częstotliwości zmian poniżej 25% dla całego projektu, tylko nieliczne mikro-usługi są często zmieniane i rozwijane. Zdecydowana większość modułów jest bardzo stabilna. W niektórych od lat nie były wykonywane żadne zmiany.

Moduły, które są najbardziej aktywnie rozwijane to: **policy**, **web**, **billing**, **commons** i **product**.

Są to równocześnie kluczowe moduły *SystemuX*. Moduły **billing**, **commons** i **product** są w bardzo dobrym stanie. Widoczna jest też poprawa jakości tych modułów w ostatnim roku.

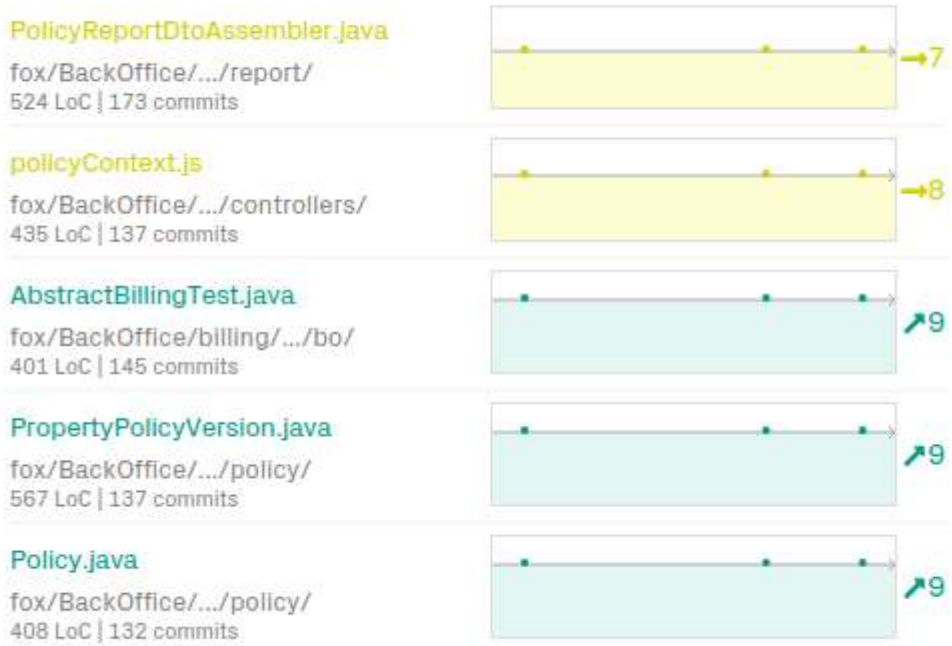
Moduły **web** i **policy** mają niższą jakość, ale biorąc pod uwagę ich złożoność - wartość 7 jest nadal bardzo dobrą.

Analiza hotspot na poziomie modułów nie wykazała istnienia istotnych zagrożeń pod kątem stanu zdrowia kodu. Analiza trendów pod kątem obniżenia jakości lub zagrożenia nie wykazała istotnych problemów. Obniżenia jakości wykryto na 2 plikach i były one poniżej 0,5 punktu.

Powyższa analiza wskazuje, że kod jest dobrej jakości. Programiści dbają o to, by eliminować najpoważniejszy dług technologiczny na bieżąco.

Poziom plików

Na poziomie pojedynczych plików hotspoty to:



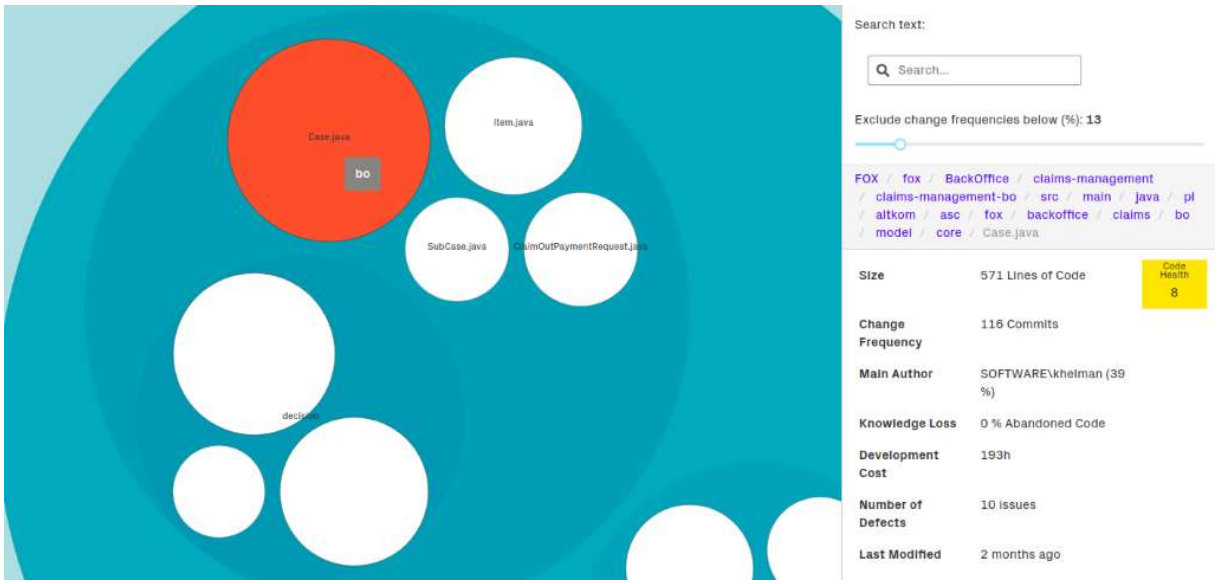
Również na poziomie plików stan zdrowia jest bardzo dobry. Widoczne są też trendy wzrostowe dla części plików. Dwa niezwykle ważne pliki to Policy i PropertyPolicyVersion, w których zaimplementowana jest obsługa operacji na polisie. Mają one bardzo dobre wskaźniki zdrowia oraz trend wzrostowy.

Na poziomie plików nie widać zagrożeń pod kątem jakości kodu.

Potencjalne obszary do poprawy

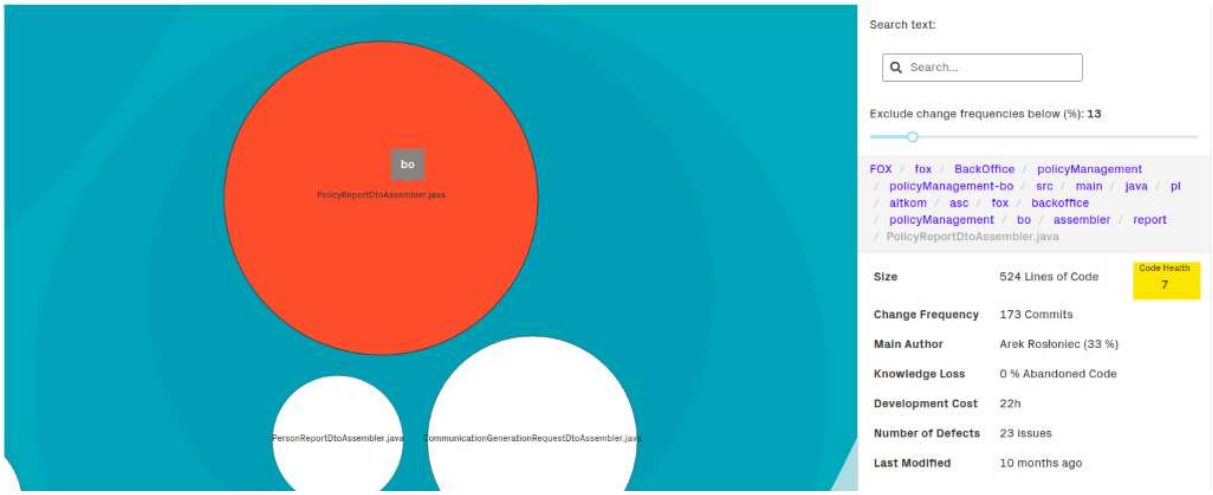
Na bazie analizy hotspot wybraliśmy następujące pliki, w których rekomendujemy wykonanie dokładniejszą analizę i rozważenie refaktoryzacji.

Pierwszy z nich to Case.java z modułu claims.

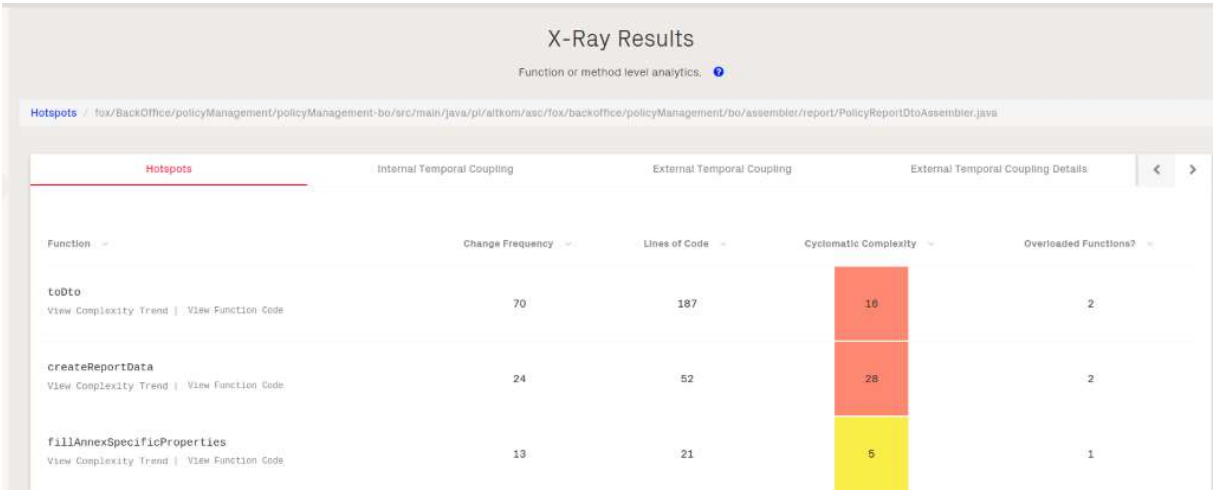


Plik ten był modyfikowany aż 118 razy. Jego wskaźnik poziomu zdrowia to 8. W chwili obecnej nie jest wymagana żadna zmiana. W przyszłości warto zwrócić uwagę na pozostawione w kodzie TODO oraz na fakt, że mamy 6 funkcji o bardzo podobnej budowie (addClaimDecision, addCostDecision, addRecourseDecision, updateClaimDecision oraz 2 inne), które być może da się zastąpić odpowiednio parametryzowaną funkcją. Konstruktor klasy ma też dużo parametrów, co może sugerować brakującą niewykrytą abstrakcję.

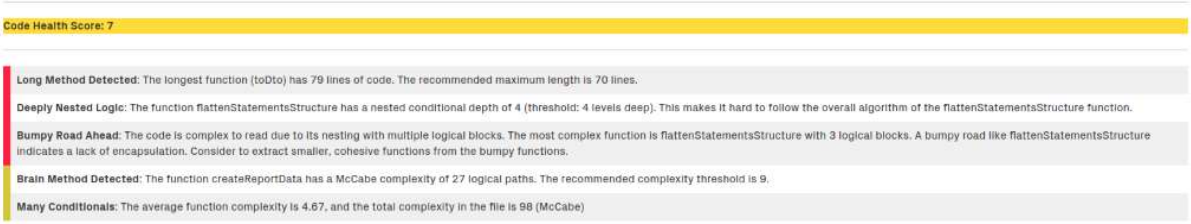
Drugi plik to PolicyReportDtoAssembler.java.



Klasa ta była modyfikowana 173 razy. Ma wskaźnik zdrowia 7, więc jest on dość wysoki. W przypadku dalszego rozwoju tej klasy warto zwrócić uwagę na: metodę flattenStatementsStructure mające 4 poziomy zagnieżdżenia warunków logicznych, zmodularyzować główną metodę toDto, która ma 187 linii i miarę Cyclomatic Complexity na poziomie 16.



Dla plików, w których wykryto problemy na poziomie analizy hotspotów, przeprowadzone zostało wirtualne code review i określono tzw. biomarkery kodu.



W analizie biomarkerów kluczowe jest odróżnienie tzw. False positive od faktycznych problemów w danym kawałku kodu. W powyższym przypadku metoda createReportData wskazana została jako bardzo skomplikowana (complexity = 27). Po przejrzeniu kodu okazuje się, że wysoka złożoność tej metody polega na użyciu switch'a z kilkunastoma enumami, gdzie większość idzie do tych samych ścieżek.

Analiza architektoniczna

SystemX złożony jest z 32 modułów. Większość z nich to mikro-usługi odpowiedzialne za poszczególne obszary biznesowe oraz aplikacje web typu Single Page Application, odpowiadające za interfejs użytkownika.

Stan zdrowia poszczególnych modułów

Moduł	Status zdrowia kodu
web	7.63
policy	7.66
billing	9.73
commons	9.32
product	9.39
claims	8.79
tariffication	7.98
aux-ledger	9.59
communication	9.79
moto-policy	10
sales-network	9.79
registry	9.19
calculations	8.86
batches	9.93
notifications	9.73
vehicle-classification	9.93
users	10
fo-api-gw	10
cas	9.8
external-data-sources	10
dictionary	10
vehicle-inventory	10
life	10
api-gw	10
promocodes	10
exchange-rates	10
party-dictionary	10
sales-configurator	9.93
tpa-api-gw	10
tasks	9.93
sales-dashboard	9.59
web-dashboard	9.86

Kolorem żółtym oznaczyliśmy moduły o jakości poniżej 8.

Jak widać większość modułów jest w bardzo dobrym stanie. Kluczowe moduły odpowiadające za interfejs użytkownika, zarządzanie polisami i liczenie składki mają jakość poniżej 8, jednak nadal jest to wartość bardzo wysoka.

W przyszłości warto monitorować status tych modułów, aby nie doszło w nich do obniżenia obecnego poziomu.

Hotspoty w modułach

Poniżej opiszemy hotspoty w kluczowych modułach oraz jedynie w zakresie, gdzie wskaźnik zdrowia jest poniżej 8. Jakość kodu w hotspotach w poszczególnych modułach jest bardzo dobra i jedynie pojedyncze pliki wymagają interwencji. Warto jednak zwrócić uwagę na te wybrane i monitorować, czy nie następuje w nich pogorszenie jakości.

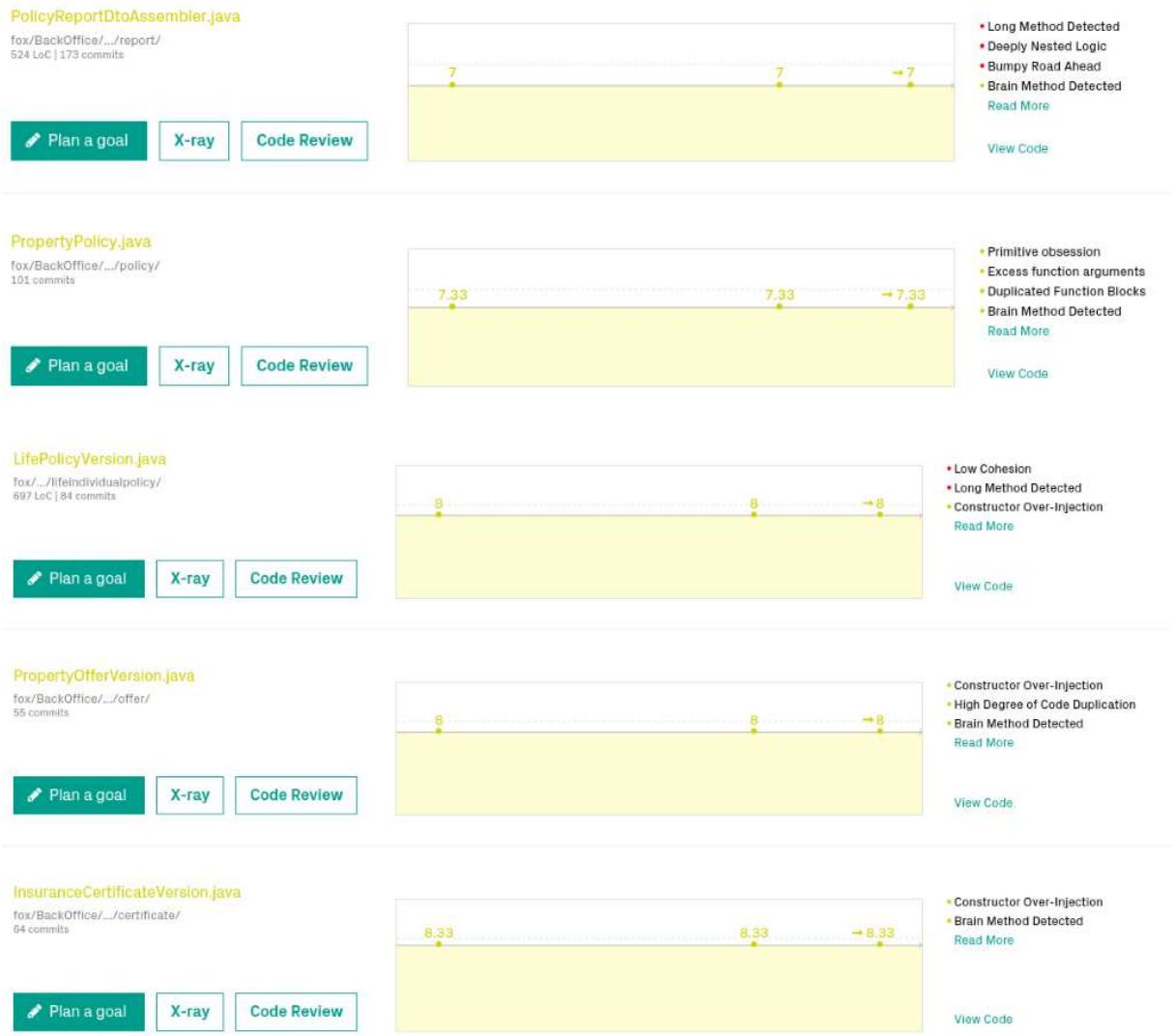
Moduł web

W module web mamy jeden plik z bardzo niskim poziomem jakości, przy relatywnie dużej (91) ilości zmian.

Dokładna analiza wykazuje wystąpienie metod z dużą ilością zagnieżdżeń logiki (4), metoda open controller ma 11 bloków utrudniających zrozumienie jej działania, występują TODO oraz funkcje mają zbyt dużą ilość parametrów (10). W przypadku dalszego planowanego rozwoju tego pliku należy zaplanować działania korygujące. Podział złożonych metod na prostsze, wydzielenie bloków kodu do metod, rozwiązanie TODO.

Moduł policy

W module polisy większość plików ma bardzo wysoką jakość. Poniżej lista tych, które są kluczowe lub widać w nich trend spadkowy.



W module tym kluczowe pliki są w dobrym stanie. Warto jednak pomyśleć, żeby w przypadku dalszego rozwoju, dodawania nowych funkcji zaplanować refaktoryzację, dzięki której usunięte zostaną problemy takie jak: długie metody, metody z dużą ilością zagnieżdżeń oraz powtarzające się bloki kodu.

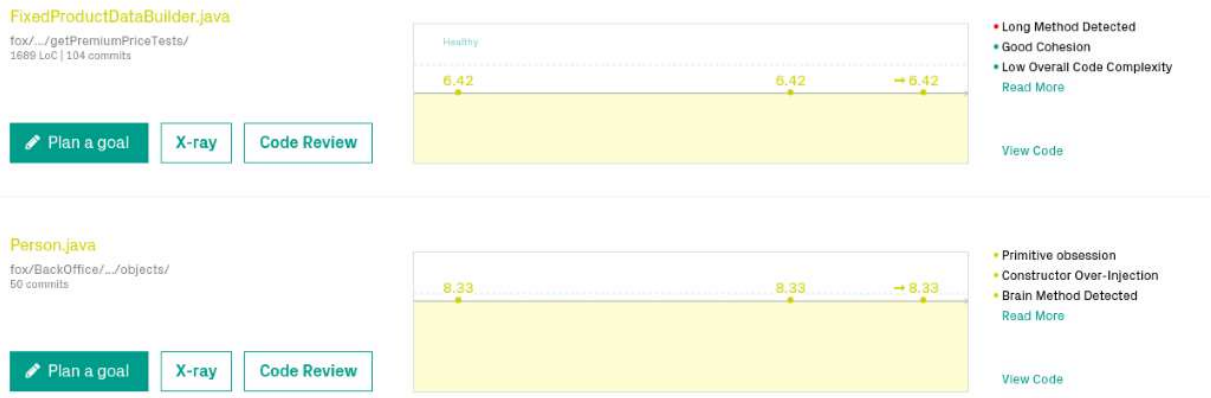
Na chwilę obecną nie są to działania konieczne.

Moduł billing

W module billing wszystkie pliki wykryte jako hotspoty mają wskaźniki zdrowia powyżej 9. W tym module nie są konieczne żadne działania naprawcze związane z jakością kodu.

Moduł taryfikacja

W module większość plików ma wskaźniki zdrowia równe lub większe 9. Jedynie 2 pliki oznaczone jako hotspot mają niższe wartości.



Pierwsza z tych klas jest klasą testową a druga jest stabilna i ma wysoki wynik 8.33. W module tym na chwilę obecną nie ma miejsc wymagających interwencji.

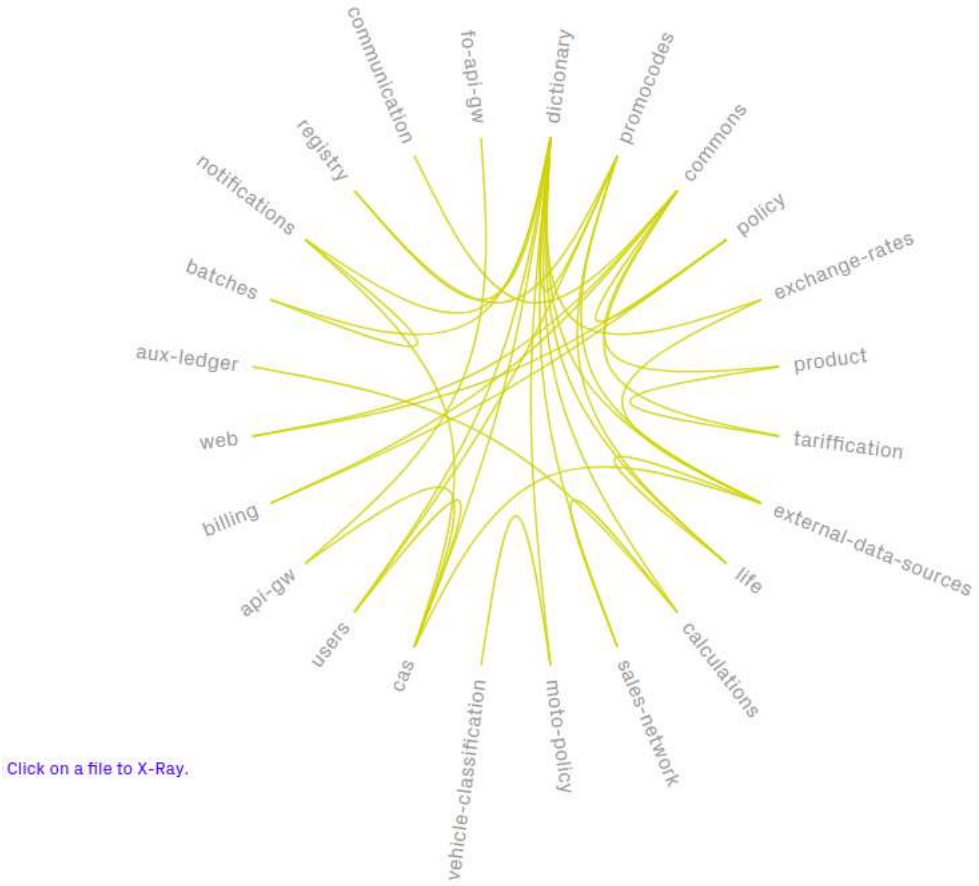
Moduł commons

W module commons wszystkie pliki wykryte jako hotspoty mają wskaźniki zdrowia powyżej 9. W tym module nie są konieczne żadne działania naprawcze związane z jakością kodu.

Moduł product

W module produkt większość hotspot'ów ma wskaźnik zdrowie większy lub równy 9. Jedynie ObjectTypeConfigurationRowParser ma niższy wskaźnik, ale jest to plik o niskiej częstotliwości zmian, a wartość 8.33 jest na tyle wysoka, że na chwilę obecną nie są konieczne żadne działania naprawcze.

Zależności między modułami



Zbadaliśmy zależności dla kluczowych modułów. W wyniku tej analizy odkryliśmy, że:

- Moduły produkt i taryfikacja są ze sobą mocno związane (36% zmian dotyczyło jednocześnie tych 2 modułów), jest to zupełnie zrozumiałe, gdyż zmiany definicji produktów wymagały zmian także w silniku kalkulacji.
- Moduł produktu jest też dość powiązany z modułem common (21%).
- Moduł polisy jest mocno związany z modułem common (45%), powiązanie to jest zbyt silne i zapewne w module common są elementy, które powinny zostać przeniesione do modułu polis.
- Moduły billing i polisy są od siebie zależne - 21% zmian dotyczyło jednocześnie tych dwóch modułów.
- Moduły sales-network i calculations są ze sobą mocno powiązane (31%)

Zależności między komponentami są stabilne i nie występuje nigdzie trend wzrostowy.

Zalecana jest analiza zależności między polisami i common. Silna korelacja między modułami może oznaczać trudności w niezależnym rozwoju poszczególnych mikroserwisów.

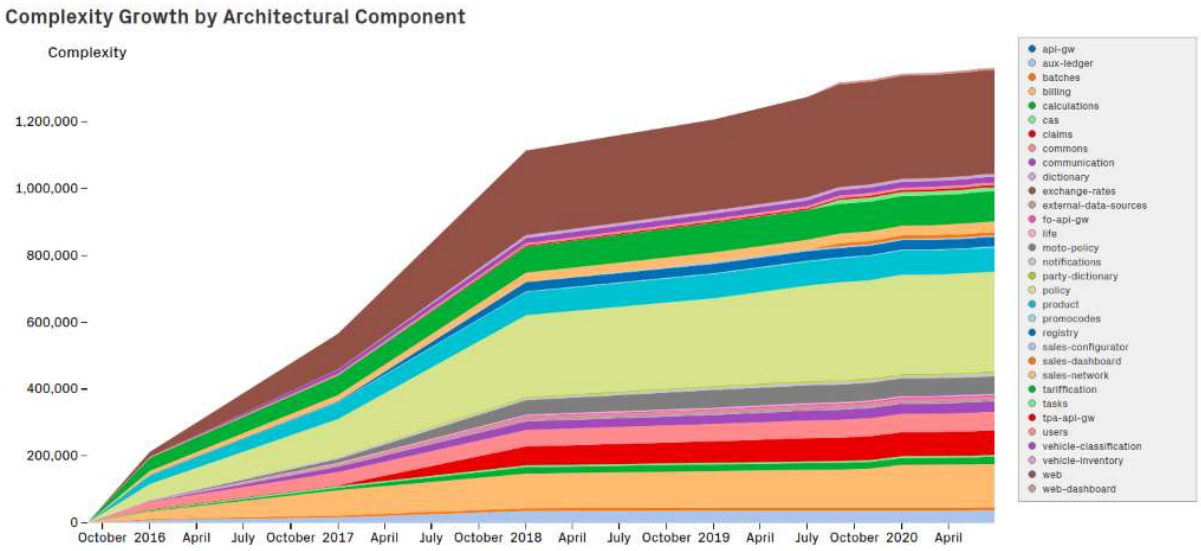
Technologie

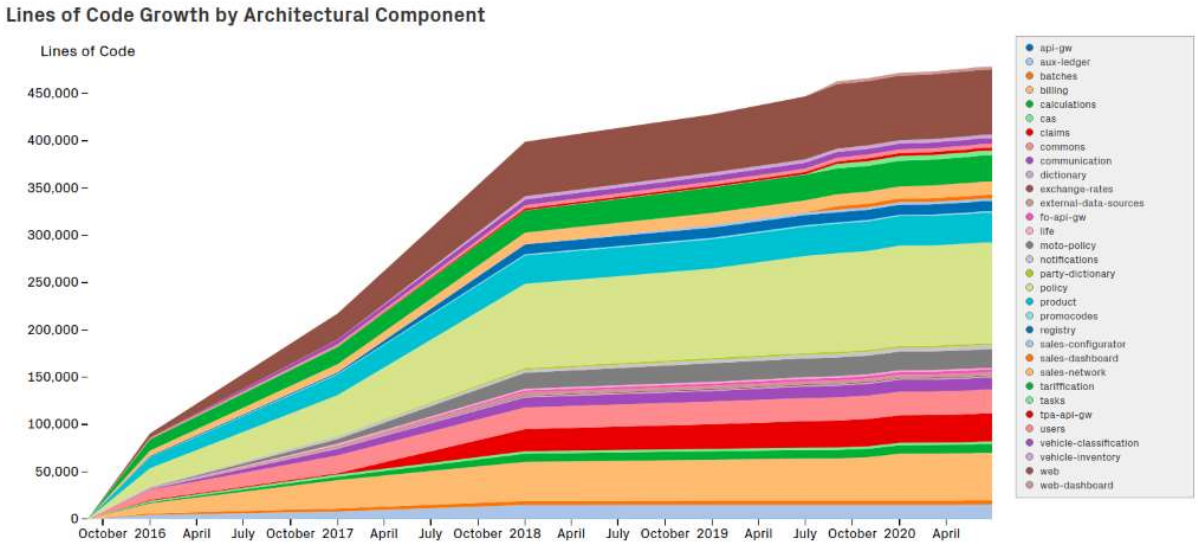
W kodzie wykorzystywane są następujące technologie:

Language	Files	Blank	Comment	Code
Java	9,913	87,943	15,536	392,344
SQL	2,337	23,582	21,456	123,462
JavaScript	1,398	15,493	20,019	105,542
JSON	161	57	0	58,705
Text	32	59	0	55,112
HTML	828	2,545	381	39,429
XML	360	2,162	1,120	30,580
Groovy	412	4,096	294	30,397
CSS	64	3,461	377	21,525
Properties	121	1,051	0	8,850
TypeScript	77	488	59	3,051
XSD	5	2	13	2,779
YAML	127	259	31	2,368
CSV	12	5	0	968
Python	19	159	68	805
TSV	1	0	0	458
DOS Batch	3	64	0	227
Markdown	10	72	0	207
Shell Script	9	50	63	201

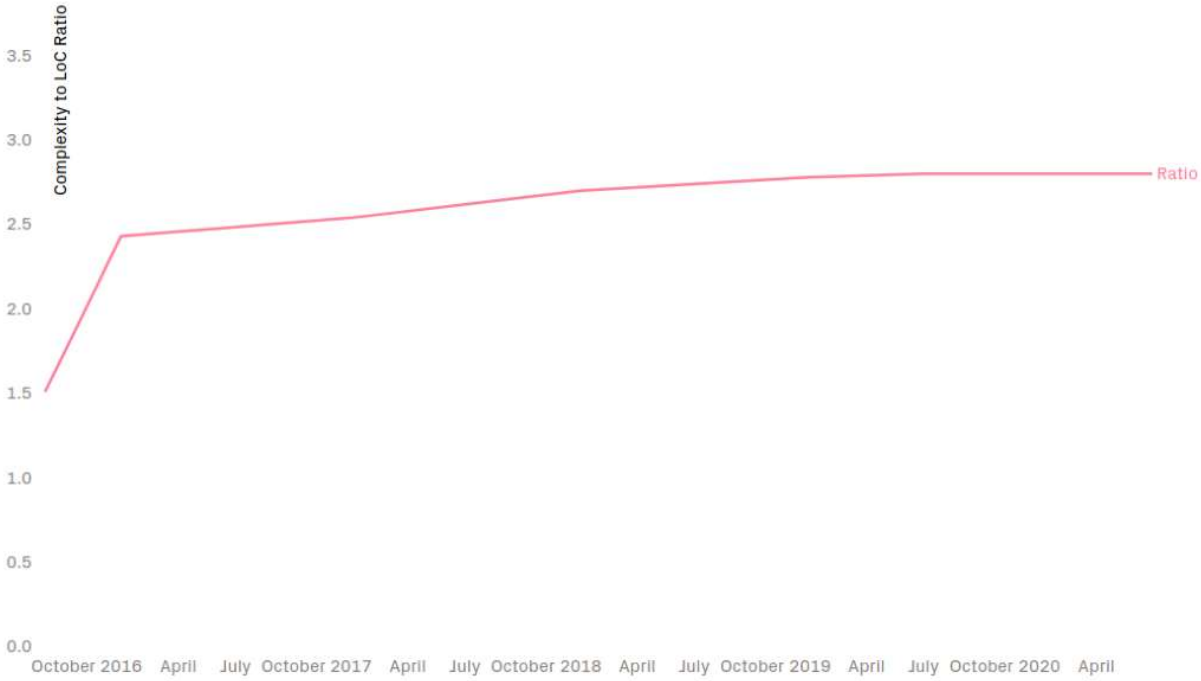
Większość kodu zgodnie z założeniami architektury to Java i Java Script. Kod SQL to definicje obiektów bazodanowych, skrypty migracyjne i poprawiające dane.

Złożoność systemu





Jak widać na wykresach wzrost ilości kodu, jak i wzrost złożoności systemu są ze sobą zbieżne, co oznacza, że system rozwijał się prawidłowo utrzymując poziom złożoności proporcjonalny do rozmiaru systemu.

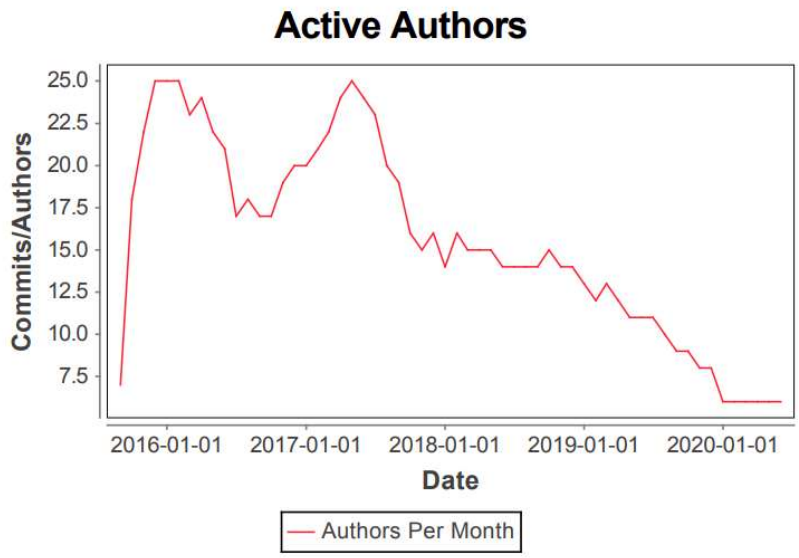


Od 2018 roku rozwój systemu wyraźnie zwolnił i kod pod względem wzrostu złożoności jest dość stabilny.

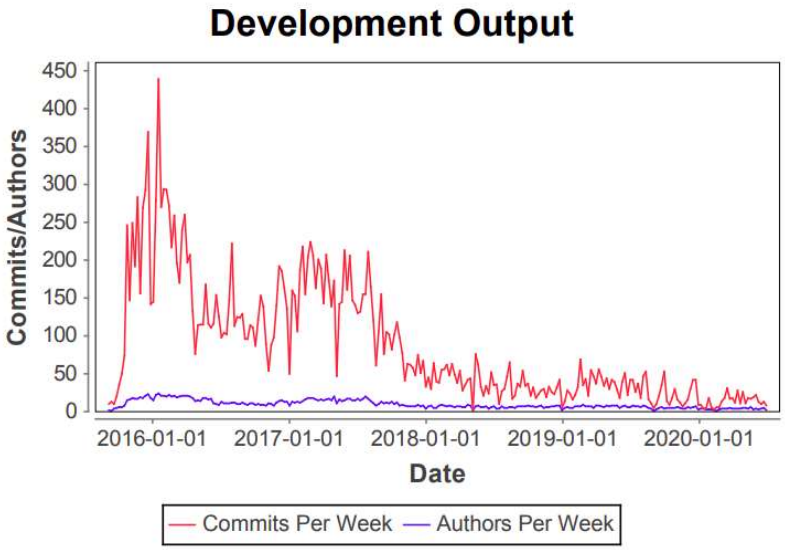
Zespół

Rozmiar i produktywność zespołu

W początkowej fazie projektu (pierwsze 1.5 roku) zespół liczył 25 osób. W ciągu ostatnich 2.5 lat rozmiar zespołu został zredukowany do 6 osób.



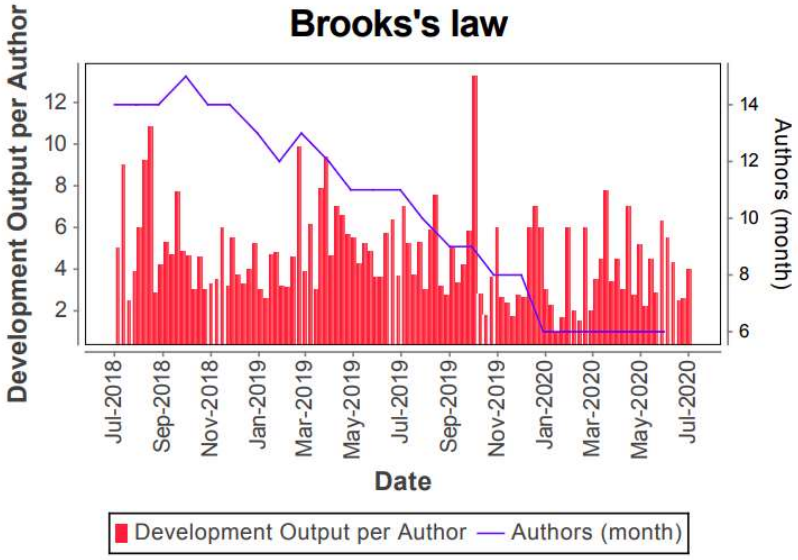
Wraz ze zmniejszeniem liczebności zespołu nastąpiło proporcjonalne zmniejszenie jego produktywności:



Przeciętna produktywność developerów (mierzona liczbą commitów) nie uległa istotnej zmianie pomimo dwukrotnego zmniejszenia rozmiaru zespołu. **Oznacza to, że skalowanie zostało przeprowadzone płynnie, bez zaburzania bieżących prac projektowych, a produktywność zmieniła się proporcjonalnie do wielkości zespołu (zgodnie z oczekiwaniem)**.

Widoczne na poniższym wykresie chwilowe wahania mogą świadczyć o występowaniu złych praktyk w procesie wytwórczym oprogramowania, np.: dużym zróżnicowaniu rozmiaru

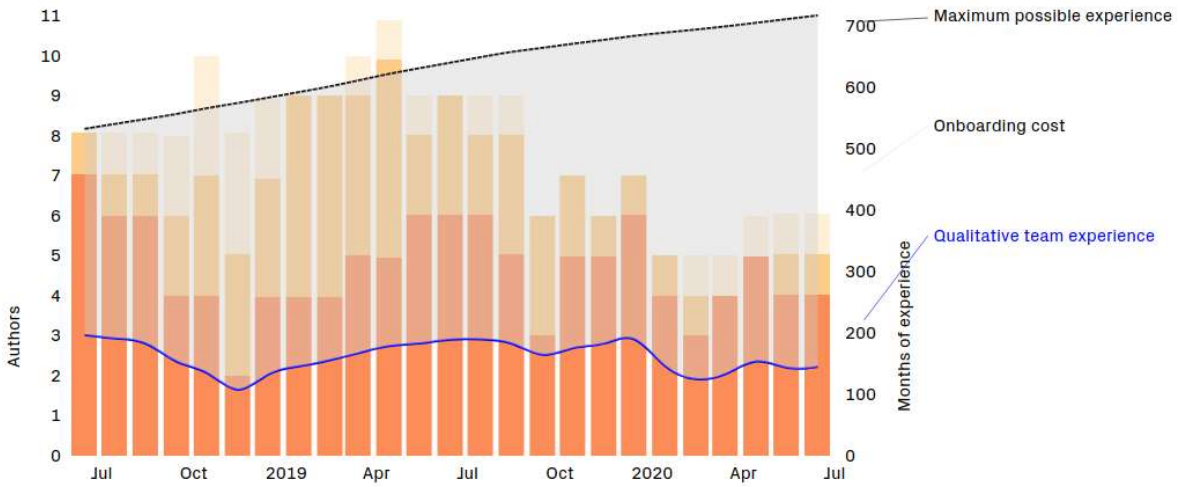
commitów lub niewłaściwej granulacji zadań. Zalecamy dokładniejszą analizę w tym obszarze.



Monthly Development Output Trend	-5% compared to previous month
Monthly Contribution Trend	-0.90 (mean) contributors compared to previous month
Weekly Development Output Trend	54% compared to last week
Weekly Contribution Trends	-3 contributors compared to last week

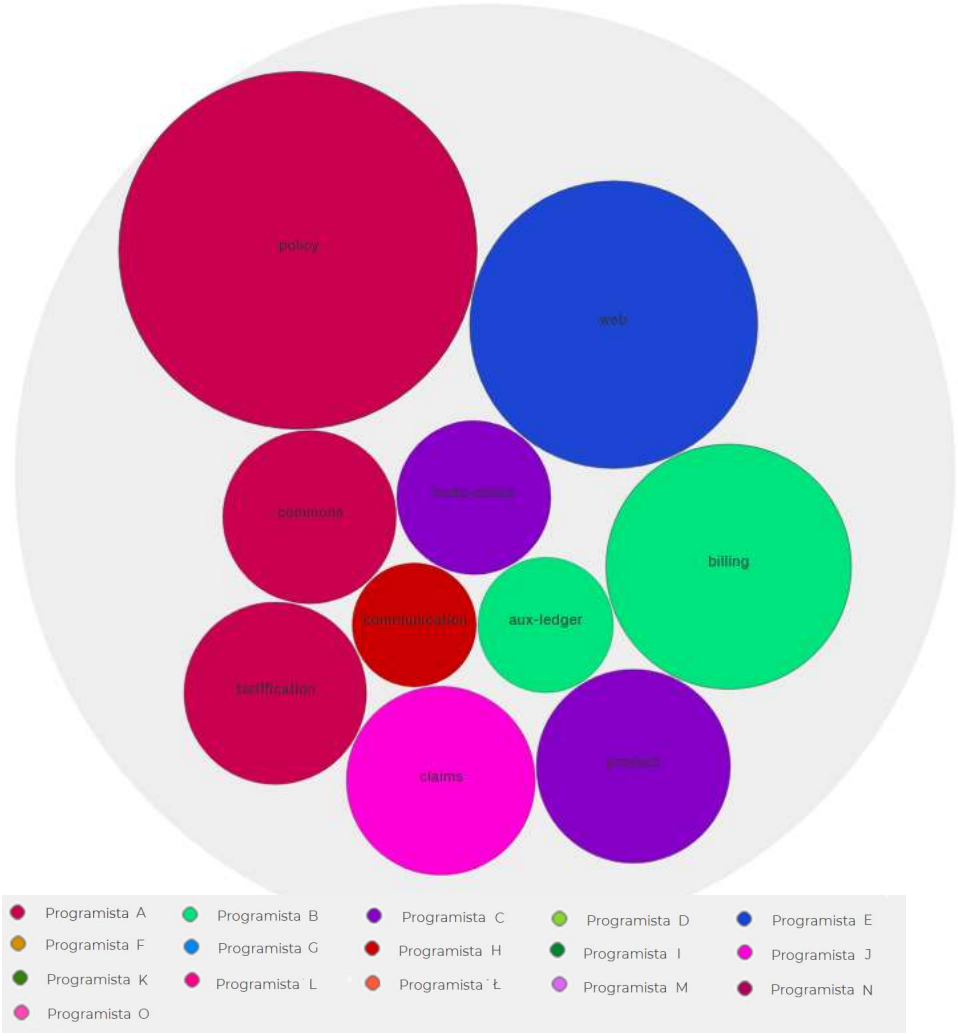
Doświadczenie i stabilność zespołu

Zespół posiada wysoki odsetek (4/6) weteranów (min 12 miesięcy w projekcie). Z drugiej strony jednak widoczna jest znacząca rotacja deweloperów, która skutkuje niestabilnością zespołu, niemożnością akumulacji doświadczenia, niskim poziomem opanowania systemu, oraz wysokim i ciągle rosnącym kosztem on-boardingu.



Developerzy

Główni autorzy / moduł:



Kod tworzony był przez 62 deweloperów. Obecnie aktywnych jest 7 deweloperów. Wykryto 53 developerów, którzy nie są już częścią zespołu.

69% kodu zostało stworzone przez deweloperów, których nie ma już w zespole projektowym. Stwarza to bardzo duże ryzyko przy rozwoju systemu, zwłaszcza, że w kluczowych modułach takich jak polisa, produkt, taryfikacja, web większość kodu stworzyły osoby, już niepracujące.

Spośród 16 głównych autorów w zespole pracują nadal *ProgramistaA* i *ProgramistaB*. Oznacza to, że dla wielu kluczowych modułów ich główni autorzy nie są już w zespole.

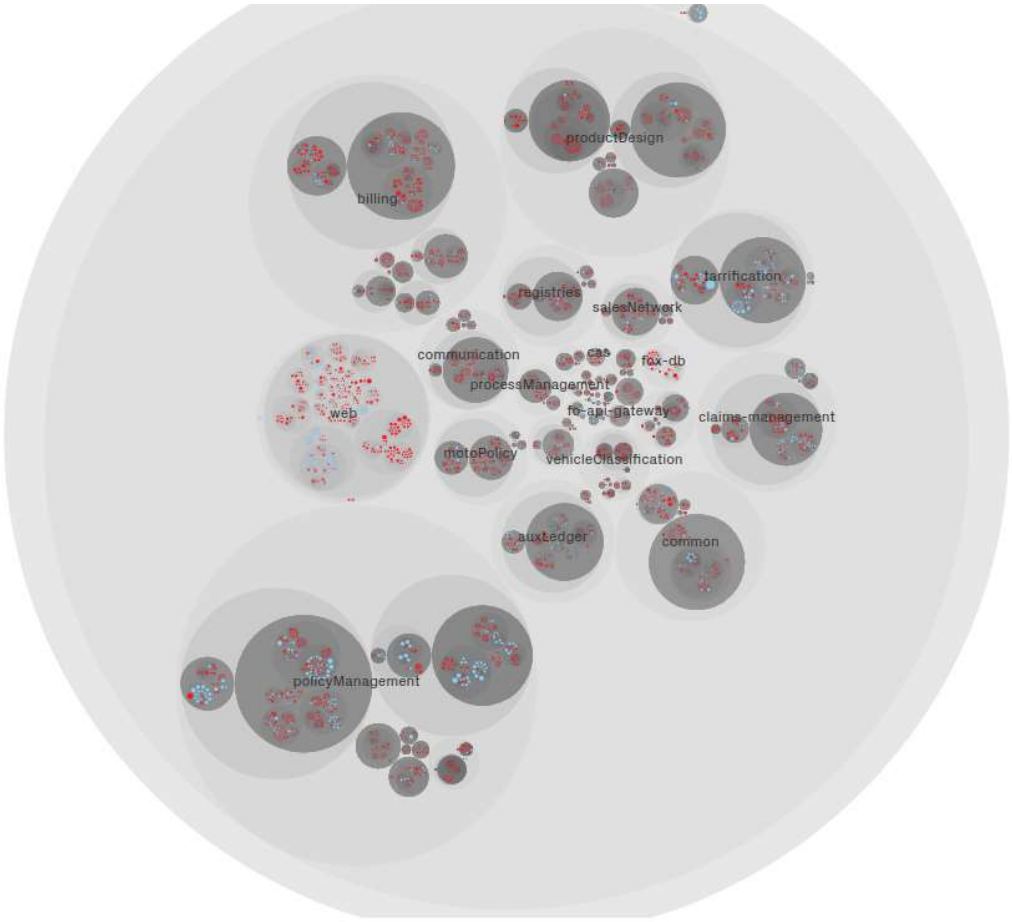
Lista deweloperów z największą ilością linii kodu

Author	Added	Deleted	Net	Revisions	Months	Last Contribution
SOFTWARE\bkoziak	202,966	40,109	162,857	286	6	2016-04-15
wojteksuwala	184,844	37,613	147,231	1543	51	2019-12-30
Tomasz Dorosz	169,726	52,402	117,324	1649	35	2018-09-26
Kamil Witkowski	124,506	33,805	90,701	142	12	2019-10-31
kozemkop	113,440	30,895	82,545	2008	35	2018-10-24
klichocki	104,641	38,395	66,246	673	27	2017-12-29
msokolowski1	115,779	53,943	61,836	1714	20	2017-07-24
Grzegorz Rutkowski	87,461	27,568	59,893	1569	55	2020-06-18
Arek Rosloniec	116,435	57,474	58,961	1963	28	2018-08-27
Dariusz Janicki	175,218	126,799	48,419	1441	23	2017-09-28
Tomasz Gryglak	44,187	393	43,794	35	1	2016-02-02
pmazurek	56,404	20,828	35,576	820	25	2017-10-31
Radosław Wosiak	39,787	5,638	34,149	171	5	2016-04-20
tsurowiec	42,744	8,711	34,033	222	24	2017-09-22
Tomasz Rutyna	57,734	24,083	33,651	984	16	2017-09-28
Tomasz Janiak	29,438	5,707	23,731	210	9	2017-05-05
Piotr Janik	24,426	9,133	15,293	179	6	2016-06-29
SOFTWARE\kheiman	13,402	1,291	12,111	93	6	2017-06-29
Arkadiusz Król	44,279	32,768	11,511	431	44	2020-04-30
akotynski	13,066	3,440	9,626	213	9	2017-08-10
arosloniec	10,057	751	9,306	27	0	2019-08-01
Piotr Smolarski	13,564	4,422	9,142	404	5	2017-11-02
SOFTWARE\wbublcz	33,490	24,483	9,007	144	19	2018-12-13

Opanowanie systemu

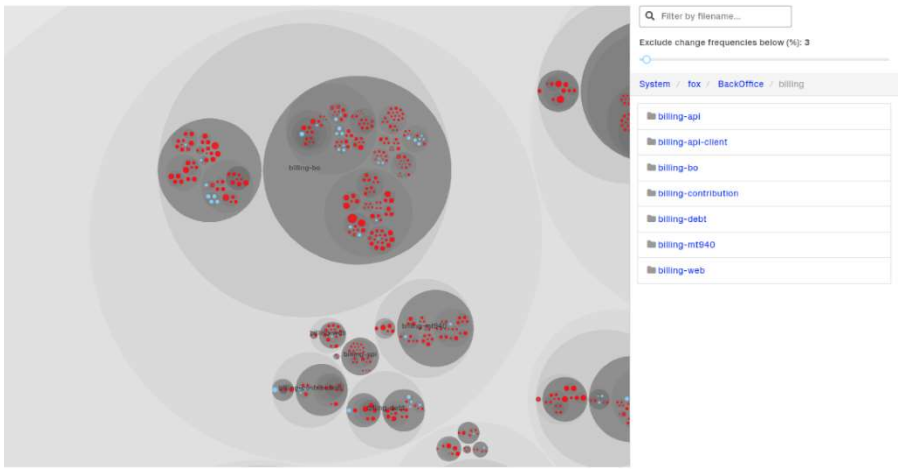
Aktualny skład zespołu charakteryzuje się niskim poziomem opanowania systemu. Wynika to z utraty wiedzy spowodowanej redukcją rozmiaru zespołu oraz rotacją developerów. Niski poziom opanowania systemu przy stosunkowo wysokim poziomie rotacji, skutkuje wysokim kosztem onboardingu nowych członków zespołu.

Dystrybucja wiedzy / braki wiedzy



Dystrybucja wiedzy w projekcie jest na złym poziomie. Większość modułów ma wysoki wskaźnik utraty wiedzy (“Knowledge Loss”). Pisana była w większości przez programistów, których nie ma już w projekcie.

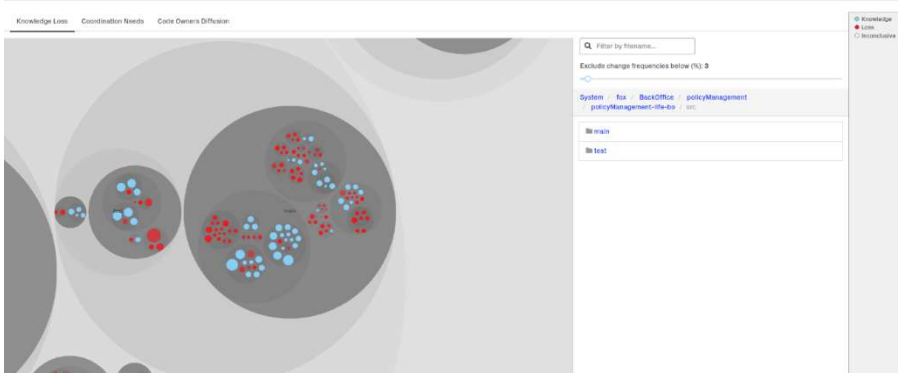
Moduły najbardziej zagrożone przez wysoki wskaźnik Knowledge Loss



Powyższy wykres przedstawia poziom wskaźnika utraty wiedzy w module billing. Moduł billing jest jednym z najważniejszych biznesowo modułów. Przez wysoki wskaźnik utraty wiedzy, błędy pojawiające się w tym module mogą być kosztowne do rozwiązania.

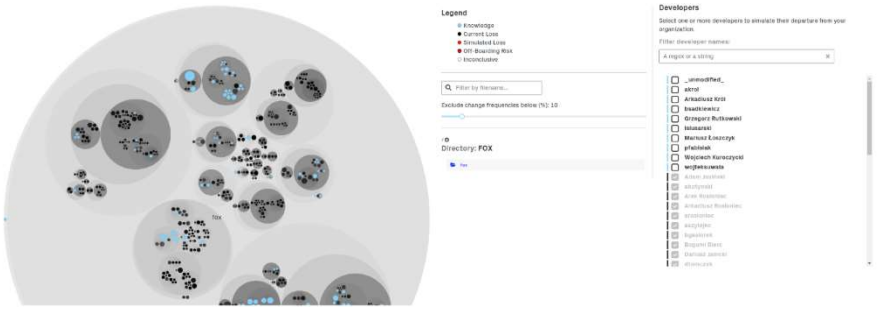
Zaleca się jak najszybsze rozpoczęcie procesu propagowania wiedzy w zespole w powyższych obszarach. Modyfikowanie kodu, którego nie programiści nie rozumieją stanowi zwiększone ryzyko projektowe.

Moduły z odpowiednio niskim wskaźnikiem Konwledge Loss



Powyższy screen przedstawia moduł policyManagement-life – jest to moduł, który wygląda zdecydowanie lepiej pod kątem dystrybucji wiedzy.

Symulacja off-boardingu



Symulacja off-boardingu pozwala przedstawić moduły, które są najbardziej zagrożone utratą wiedzy w przypadku odejścia jednego z aktywnych programistów.

Z powyższego obrazka można wywnioskować, że z powodu swojej rozpiętości i czasu rozwoju (ponad 5 lat) posiada dużo części, które już teraz uznawane są za “Current Loss”, czyli części, w których aktualnie pracujący programiści nie wykonywali zmian.

W przypadku odejścia *ProgramistyB*, spora część modułów jest narażona na utratę wiedzy. Są to: **claim-management, billing** oraz moduły **związane z registerCase oraz claims.**

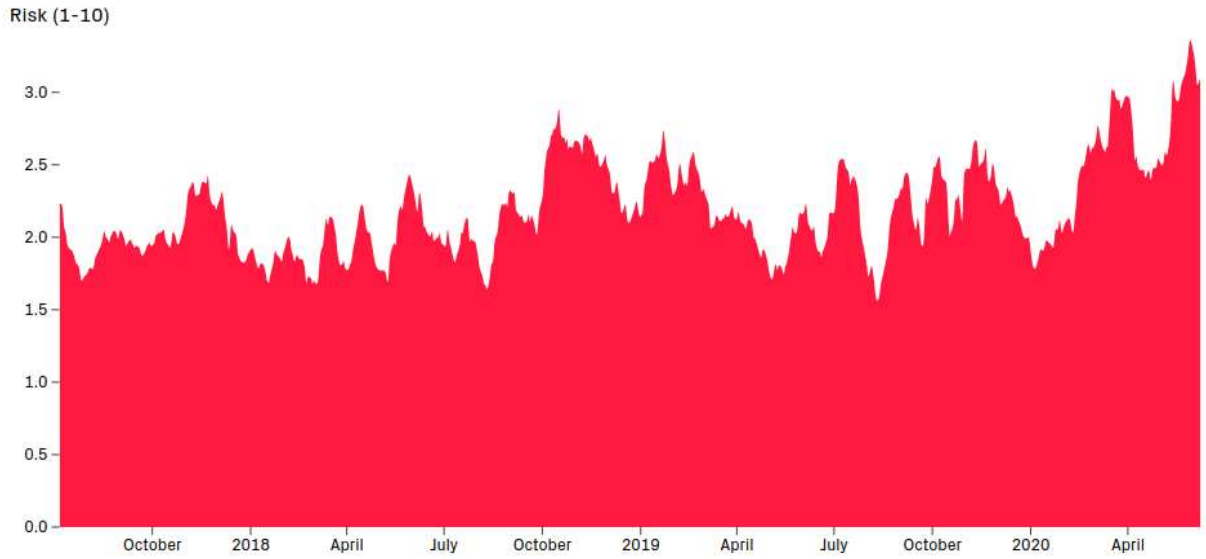
Proces wytwórczy

Czas realizacji

Niekompletne informacje w systemie JIRA uniemożliwiają przeprowadzenie automatycznej analizy czasu realizacji zadań

Ryzyko zmian

Zmiany wprowadzane do systemu w procesie wytwarzania oprogramowania charakteryzują się niskim poziomem ryzyka. Wynika to z doświadczenia developerów (patrz rozdział "Doświadczenie i stabilność zespołu"), granulacji zmian oraz modularności systemu.

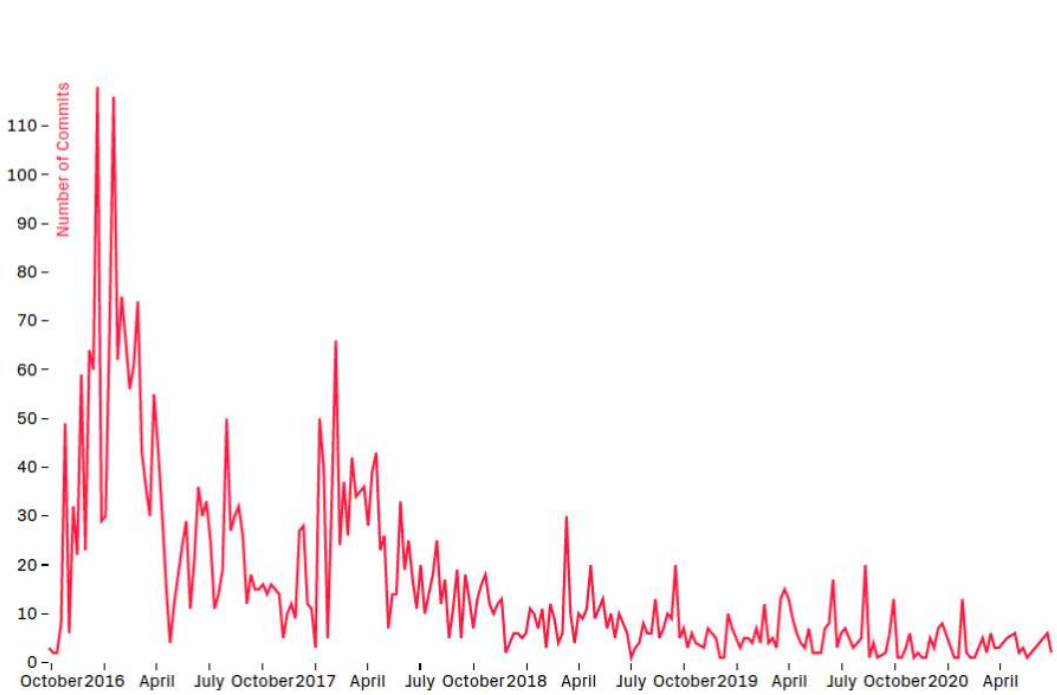


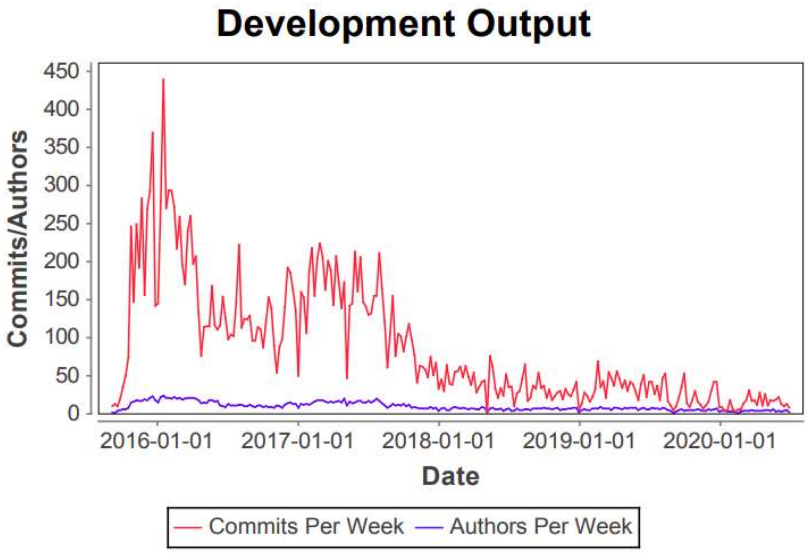
Praca planowana vs nieplanowana

Niekompletne informacje w systemie JIRA uniemożliwiają przeprowadzenie automatycznej analizy rozkładu pracochłonności zadań.

Rodzaj realizowanych zadań

Liczba commitów dotyczących naprawy błędów utrzymuje się na poziomie około 20 procent i charakteryzuje stosunkowo niewielkimi wahaniami. Świadczy to o poprawnym zarządzaniu jakością w projekcie i pokrywa z wnioskami wypływającymi z analizy ryzyka zmian (patrz rozdział "Ryzyko zmian").



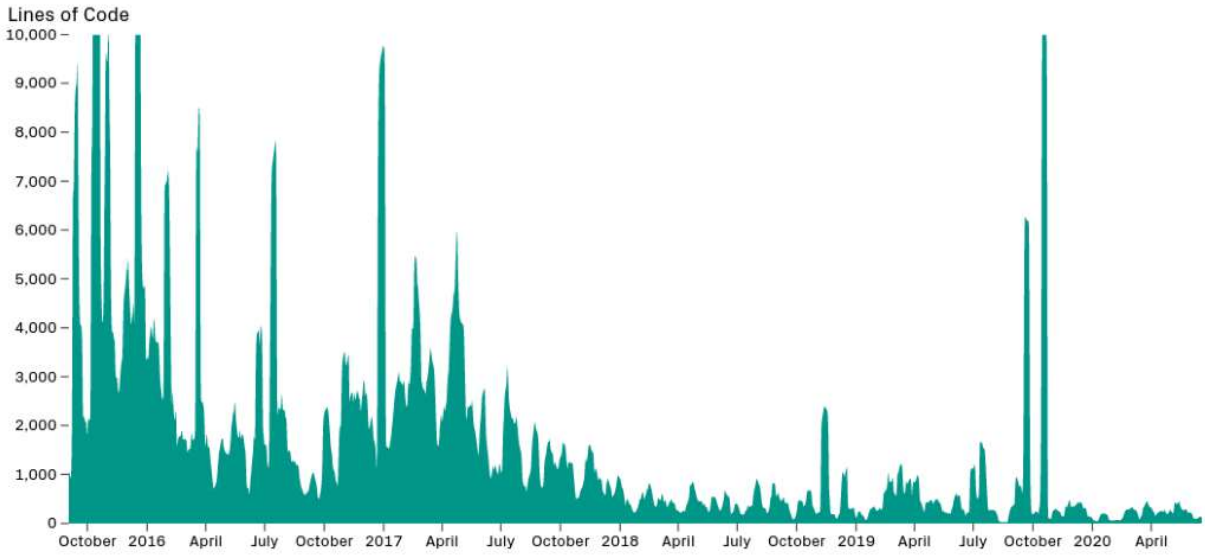


Koszt tworzenia oprogramowania

Niekompletne informacje w systemie JIRA uniemożliwiają przeprowadzenie analizy kosztów.

Code churn

Liczba dodanych linii kodu:



Liczba usuniętych linii kodu: