

1. O projekcie

Cel projektu

DePress (Defect Prediction in Software Systems) jest frameworkiem bazującym na projekcie KNIME. Pozwala na analizę danych dotyczących oprogramowania za pomocą węzłów przedstawianych graficznie. Celem naszego projektu jest zmiana struktury projektu DePress oraz przetestowanie jego węzłów w taki sposób, aby zostały one zaakceptowane przez zespół KNIME, jako integralna część projektu.

Wstępna analiza

Analiza węzłów

Podczas pierwszego etapu pracy nad projektem zespół zapoznał się z możliwościami, jakie oferują węzły projektu DePress. Poniższa tabela przedstawia krótko charakterystykę węzłów tego projektu.

Węzeł	Wejście	Wyjście
Git Offline	Plik XML wygenerowany za pomocą polecenia: <code>git log --pretty=format:"%H%n%ct%n%an%n%B%n%H" --raw --no-merges --abbrev=40</code>	Data Change History -> Tabela z kolumnami: Class, Extension, Author, Action, Message, Path, Date, CommitID
Git Online	Adres URL repozytorium, login oraz hasło	Data Change History -> Tabela z kolumnami: Class, Extension, Author, Action, Message, Path, Date, CommitID
SVN Offline	Plik XML wygenerowany za pomocą polecenia: <code>svn log --xml --verbose --use-merge-history repository_url > result.xml</code>	Change History Data -> Tabela z kolumnami: Class, Extension, Author, Action, Message, Path, Date, CommitID
SVN Online	Adres URL repozytorium, login oraz hasło	Change History Data -> Tabela z kolumnami: Class, Extension, Author, Action, Message, Path, Date, CommitID
Bugzilla Offline	Plik XML dla danego projektu w systemie Bugzilla	Issue Tracking System Data Table -> Tabela z danymi o issue: ID, Created, Resolved, Updated, Time Estimate, Time Spent, Status, Type, Resolution, Version itd.
Jira Offline	Plik XML uzyskany za pomocą przełączenia się na widok XML na stronie analizowanego projektu w systemie JIRA	Issue Tracking System Data Table -> Tabela z danymi o issue: ID, Created, Resolved, Updated, Time Estimate, Time Spent, Status, Type, Resolution, Version itd.
Encryption	Port wejściowy: źródło danych (na przykład węzeł CSV Reader) Parametry wejściowe: kolumny, które mają być zaszyfrowane (za pomocą podanego hasła)	Tabela z takimi samymi nazwami kolumn jak na wejściu, ale wartościami zaszyfrowanymi dla podanych na wejściu kolumn
Decryption	Port wejściowy: źródło danych	Tabela z takimi samymi nazwami kolumn jak na wejściu, ale

	Parametry wejściowe: kolumny, które mają być odszyfrowane (za pomocą podanego hasła).	wartościami odszyfrowanymi dla podanych na wejściu kolumn. Algorytm szyfrowania jest symetryczny (możemy odszyfrować dane za pomocą hasła użytego do szyfrowania)
Marker Parser	Port wejściowy: węzeł SCM (np. GIT Online) Parametry wejściowe: wyrażenie regularne	Change history data and markers -> Tabela z kolumnami: Class, Extension, Author, Action, Message, Path, Date, CommitID, Marker
Issues per Artefact	Porty wejściowe: węzeł SCM (np. GIT Online) oraz węzeł ITS (np. JIRA Online)	Metric results -> Tabela z kolumnami: RowID, Issues, NumberOfIssues, NumberOfUniqueIssues, HasIssues
Jacoco Adapter	Plik konfiguracyjny XML	Tabela z kolumnami: LineCoverage, InstructionsCoverage, BrancheCoverage, CpmplexityCoverage, MethodCoverage, ClassCoverage
CKJM	Plik XML wygenerowany przez zmodyfikowaną wersję CKJM (http://gromit.iia.pwr.wroc.pl/p_in_f/ckjm), dopiero wygenerowany XML przez anta był akceptowany przez węzeł	Tabela: kolumny – poszczególne metryki, wiersze – nazwy analizowanych klas
CheckStyleAdapter	Z opisu wynika, że plik XML wygenerowany przez Checkstyle (problem z przetestowaniem)	Tabela z kolumnami: Class, LineNumber, ColumnNumber, SeverityType, MessageText, SourcePlace

Analiza struktury projektów KNIME i DEPRESS

W odpowiedzi zwrotnej od społeczności KNIME zawarta została uwaga dotycząca aktualnej struktury projektu DePress. Obecnie dla każdego węzła zdefiniowany jest oddzielny projekt. Węzły kompilowane są niezależnie i dla każdego z nich generowany jest plik plugin.jar. Ponadto każdy z tych projektów odwołuje się do bazowego projektu (ic-depress-base - depressbase.jar), który odpowiedzialny jest za definicję powtarzających się klas oraz stworzenie struktury katalogów. Taka struktura może nieść za sobą ograniczenia czasowe związane z czasem ładowania węzłów. Zgodnie z uwagą, struktura zostanie zmieniona. W ramach zadań projektowych utworzymy jeden główny projekt, w którym znajdować się będą definicje klas bazowych wykorzystywanych przez węzły. W ramach iteracji będziemy stopniowo dodawać do projektu kolejne węzły w zależności od ich priorytetu. Przeniesione węzły będą stanowić jeden projekt. Zawarty w tym projekcie plik plugin.xml będzie odpowiedzialny za stworzenie struktury katalogów oraz będzie wskazywał definicję wszystkich występujących węzłów. W konsekwencji otrzymamy tylko jeden, ale obszerniejszy plugin, co powinno pozwolić uchronić projekt przed zwiększonym czasem ładowania węzłów.

2. Backlog zadań

Każde zadanie polega na przeniesieniu kolejnego węzła projektu DePress do wspólnej wtyczki. Zbiór planowanych zadań, którymi są modyfikacje węzłów projektu Depress oraz przypisany im priorytet przedstawia poniższa tabela.

Zadanie	Priorytet
SCM:Git SCM Offline	6
SCM:SVN SCM Offline	6
ITS:Bugzilla Offline	5
ITS:Jira Offline	5
Data:Encryption	4
Data:Decryption	4
Matcher:Marker Parser	3
MG:Issues per Artefact	2
Mr:Jacoco Adapter	1
Mr:PiTest	1
Mr:SourceCrawler	1

Im wyższy priorytet tym zadanie ważniejsze.

3. Metodyka pracy

Wybrana metodyka

Proponowaną przez nas metodyką pracy nad projektem jest Kanban, głównie ze względu na trudną do czasowego oszacowania pracę na styku naszego zespołu oraz zespołu projektu KNIME.

Kanban nie narzuca sztywnych ram czasowych realizacji poszczególnych czynności (faz procesu), w rezultacie powodzenie projektu w dużej mierze opiera się na zaangażowaniu członków zespołu projektowego. Nie chcemy także doprowadzić do zbędnych operacji po stronie naszego zespołu, jeśli Team KNIME będzie w innym tempie pracować. Metodyka Kanban zakłada odejście od podejścia iteracyjnego, praca zespołu toczy się w sposób ciągły.

Nie występują także w Kanbanie harmonogramy realizowania poszczególnych czynności (faz procesu), zostały one zastąpione limitami pracy w toku (WIP). Dla naszego procesu i jego poszczególnych faz określono pewne ograniczenia, które szczegółowo opisano w podpunkcie [Ograniczenia nałożone na proces](#).

Analiza całego cyklu procesu poszczególnych zadań, da nam informacje o „wąskim gardle” procesu, czyli jaki etap zadania (faza procesu) zajmował najwięcej czasu, na co będziemy mogli zareagować.

Aby odpowiednio przebiegała praca w metodyce Kanban najważniejszy jest odpowiedni przepływ informacji pomiędzy członkami zespołu. Tablica Kanban, która jest nieodłącznym elementem tej metodyki ma za zadanie informować o aktualnym stanie przebiegu prac nad zadaniami, członkowie zespołu są zobowiązani do bieżącej aktualizacji tablicy. W tej roli podczas naszego projektu będzie wykorzystywane narzędzie Kanbanize, o którym więcej w podpunkcie [Narzędzie wspomagające metodykę](#).

Fazy procesu

Fazy procesu nad którym bezpośrednio będzie pracował nasz zespół to fazy Refactoring, Testing, Code review & functional testing. Fazy związane z wdrożeniem, tj. Deploy @ depress i Deploy @ knime, przede wszystkim związane są z wizualizacją przebiegu wdrożenia naszej modyfikacji projektu depress do projektu KNIME.

Faza	Opis
1. Refactoring	Refaktoryzacja polegająca na przeniesieniu klas/pakietów węzła do nowej wtyczki. W tym refaktoryzacja kodu źródłowego w celu zwiększenia czytelności, m.in. z użyciem wtyczki checkstyle (przyjęty standard kodowania Suna).
2. Testing	Faza polegająca na uporządkowaniu i/lub napisaniu testów jednostkowych do kluczowych klas wtyczki
3. Code review & functional testing	Testowanie funkcjonalne poprawności wtyczki, inspekcja kodu źródłowego, w tym drobne korekty mające na celu poprawę czytelności kodu źródłowego.
4. Deploy @ depress	Faza polega na udostępnieniu zmodyfikowanej wersji depressa do forka projektu ic-depress na GitHubie. Kończy się zatwierdzeniem zmian naszego zespołu przez Marka Majchrzaka do brancha knime_contribution projektu ic-depress na GitHubie.
5. Deploy @ knime	Faza polega na weryfikacji, przetestowaniu i zatwierdzeniu zmodyfikowanej wersji depressa przez KNIME Team.

Role w zespole

Metodyka kanban nie zakłada ról, w przeciwieństwie np. do metodyki Agile, w której wyróżniamy takie role jak: product owner czy scrum master.

Aby uniknąć przepracowania, długotrwałego oddziaływania stresu tylko przez część naszego zespołu, a także by każda osoba z naszego zespołu miała odpowiednią wiedzę o całym procesie i strukturze projektu zdecydowano się na wprowadzenie rotacyjnych ról na fazach procesu: Refactoring, Testing, Code review & functional testing.

Żadna osoba z zespołu nie ma na stałe przypisanej roli do fazy procesu (Refactoring, Testing, Code review & functional testing)

Role w zespole przydzielane są w sposób rotacyjny. Początkowo dwie osoby zostają przydzielone do fazy refraktoryzacji, a dwie pozostałe do fazy testowania oraz Code review & functional testing. W kolejnym etapie następuje wymiana jednej osoby (która była w danej fazie najdłużej) z fazy refactoringu na osobę z pozostałych dwóch faz (na takich samych zasadach jak powyżej). Dzięki temu osoba, która przechodzi do nowej fazy współdziała z osobą, która jest zapoznana z tematem. Rotacje odbywają się w celu zrównoważenia pracy dla każdego członka zespołu.

Podczas fazy refraktoryzacji dwie osoby zajmują się dwoma zadaniami (każda osoba osobnym zadaniem). Natomiast do fazy testowania każdej z pozostałych dwóch osób przydzielone jest jedno zadanie w fazie testowania, po czym następuje wymiana tych zadań dla fazy code review & functional testing w celu ostatecznej weryfikacji poprawności wtyczki.

Po ukończeniu zadań wtyczka przechodzi przez fazy deploy aż do końcowej akceptacji.

Dla sklaryfikowania powyższego opisu podamy przykład działania na podstawie naszego pierwszego sprintu.

Łukasz oraz Wojciech zostają przydzieleni do fazy refraktoryzacji, w której mamy za zadanie sprawdzenie i przeniesienia wtyczek do GIT'a oraz SVN'a. Po ukończeniu tych zadań (każda z osób otrzymała po jednej wtyczce). Po ukończeniu fazy refraktoryzacji Adrian oraz Paulina otrzymują za zadanie testowania po jednej spośród tych dwóch wtyczek (np. Adrian testuje wtyczkę GIT, a Paulina wtyczkę SVN) następnie po ukończeniu fazy testowania następuje wymiana wtyczek w fazie code review & functional testing (dla powyższego przykładu w fazie code review & functional testing Adrian będzie się zajmował wtyczką SVN, a Paulina GIT). Po ukończeniu wszystkich zadań następuje wymiana osób (np. Adrian przechodzi do fazy refraktoryzacji, Łukasz do fazy testowania i code review & functional testing - po kolejnym etapie Wojciech przeniesie się do fazy testowania i code review & functional testing, a Paulina przejdzie do fazy refraktoryzacji itd.).

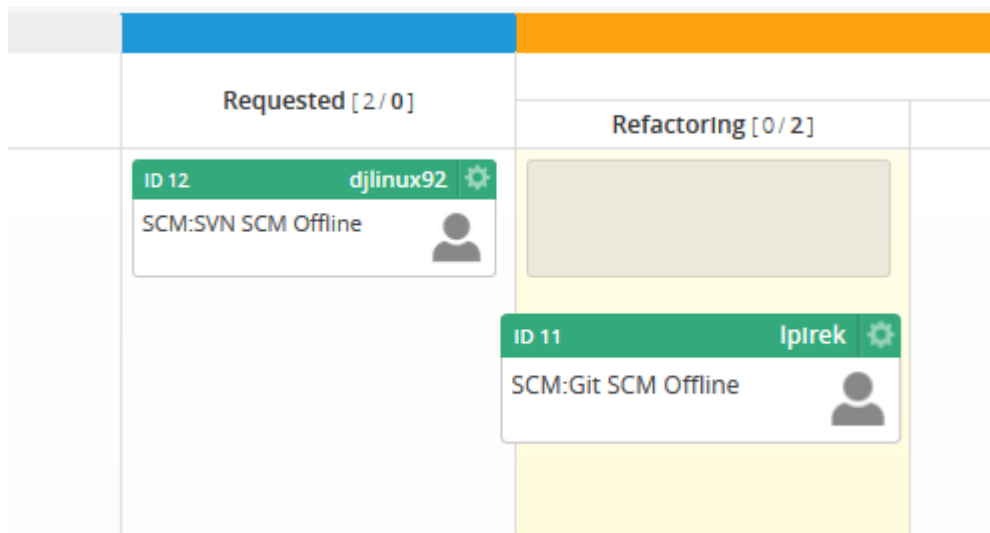
Ograniczenia nałożone na proces

1. W fazie refactoring co najwyżej dwa zadania mogą być przetwarzane.
2. Nad jednym zadaniem w fazie refactoring mogą pracować co najwyżej jedna osoby.
3. W fazie testing co najwyżej dwa zadanie mogą być przetwarzane.
4. Nad jednym zadaniem w fazie testing może pracować tylko jedna osoba.
5. W fazie code review & functional testing co najwyżej dwa zadania mogą być przetwarzane.
6. Nad jednym zadaniem w fazie code review & functional testing może pracować tylko jedna osoba.
7. Co najwyżej dwa zadania mogą znajdować się w fazie deploy @ depress.
8. Co najwyżej dwa zadania mogą znajdować się w fazie deploy @ knime.

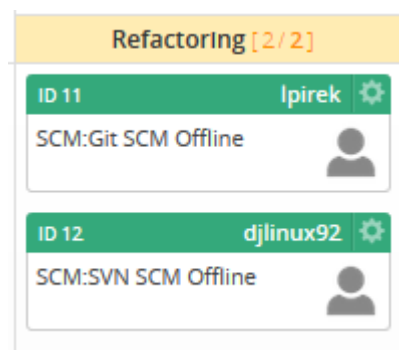
Narzędzie wspomagające metodykę

Narzędziem wspomagającym pracę z wybraną metodyką Kanban przez nasz zespół jest [Kanbanize](#)

Praca z tym narzędziem zostanie rozpoczęta przez utworzenie tablicy Kanban z fazami procesu opisanymi wcześniej, które są widoczne na poniższym zrzucie ekranu.



Rysunek 3 Przypisanie zadań do osób i intuicyjne przenoszenie zadań do kolejnych faz



Rysunek 4 Podpowiedź systemu o osiągnięciu limitu dla fazy procesu Refactoring

User WIP settings

Selected users : dlinux92

Currently assigned tasks in all boards

Global limit

1

The maximum number of tasks allowed for a person on a global level.

☒ Backlog

0

The number of allowed tasks in Backlog.

☒ Requested

0

The number of allowed tasks in the Requested area of the board.

☒ In Progress

1

The number of allowed tasks in the In Progress area of the board.

☒ Done

0

The number of allowed tasks in the Done area of the board.



☒ Archive

0

The number of allowed tasks in the temporary Archive.

Rysunek 5 Narzędzie umożliwia utworzenie limitów związanych z liczbą zadań do których jesteśmy przypisani dla poszczególnych faz.

SCM:Git SCM Offline

 | 

Task ID: 11	Subtasks	Comments	History	Metrics	Links
-------------	----------	----------	---------	---------	-------

☒ Include time logged for subtasks ☐ Include time logged for the task children

Column name	Elapsed time	Logged time
▼ Backlog	13 min, 4 sec	0.00 h.
^ Requested	33 sec	0.00 h.
^ Requested	33 sec	0.00 h.
^ In Progress	7 min, 50 sec	0.00 h.
^ In Progress	7 min, 51 sec	0.00 h.
▼ Refactoring	7 min, 18 sec	0.00 h.
▼ Testing	34 sec	0.00 h.
▼ Code review	0 sec	0.00 h.
▼ Deploy	0 sec	0.00 h.
▼ Done	0 sec	0.00 h.

Rysunek 6 Poza historią działań związanych z zadaniem narzędzie przedstawia metryki, które mogą się przydać do zlokalizowania "wąskiego gardła" procesu

4. Wykorzystywane narzędzia

Eclipse

Do pracy nad projektem wykorzystujemy KNIME SDK ze strony projektu KNIME tj.:

- Eclipse w wersji 3.7.2,
- Knime w wersji 2.12.1

Checkstyle

Dla zapewnienia pewnego standardu czytelności kodu źródłowego zdecydowano się na zastosowanie konfiguracji checkstyle zgodnej z konwencją Sun. Konwencja stylu, do której cały zespół będzie się stosował znajduje się w pliku `sun_checks.xml`, w głównym katalogu repozytorium. Zespół bierze pod uwagę dostosowanie konwencji do stylu kodu projektu KNIME.

Git

Do pracy nad reorganizacją węzłów i wtyczek projektu depress postanowiono utworzyć kopie repozytorium projektu na serwerze bitbucket w następujący sposób:

```
git clone -b knime_contribution --single-branch https://github.com/w0st/ic-depress.git
git checkout -b agile
git remote add bb https://wojciech-tepniak@bitbucket.org/agile15_4/agile15_4.git
git remote -v
git push -u bb agile
```

Gdzie repozytorium `https://github.com/w0st/ic-depress.git` to fork repozytorium `ic-depress`.

W fazach projektu: *Refactoring*, *Testing*, *Code review* & *functional testing* praca zespołu odbywa się tylko na repozytorium bitbucket: `agile15_4`

Na potrzeby poszczególnych członków zespołu lub podgrup zespołu system Git udostępnia możliwość lokalnego wprowadzania rewizji, wycofywania zmian jak i dodatkowych rozgałęzień (branches) co jest niewątpliwie atutem narzędzia kontroli wersji jakim jest Git.

Po każdej skończonej fazie *Code review* & *functional testing*, a w ramach fazy *Deploy @ depress* wersja z bitbucket `agile15_4` trafia do forka na githubie (`https://github.com/w0st/ic-depress.git`). Kolejny krok to wysłanie pull-request do `knime_contribution@depress`.

5. Inne uwagi

Pomysł na wtyczkę TFS

Zadaniem wtyczki miałyby być zaimportowanie logów z systemu TFS w wersji od 2008 wzwyż do systemu DePress. Polegało by ono na odpowiednim przetworzeniu generowanego, przez TFS, pliku logu w formacie `.xls` do formatu danych akceptowanych przez węzły KNIME. Działanie wtyczki byłoby analogiczne do wtyczki SCM:GIT

Przy sprawnej integracji projektu DePress do oficjalnej wersji KNIME rozważamy podjęcie się wykonania tej wtyczki.