BAB II

TINJAUAN PUSTAKA

Pada bab ini diberikan definisi-definisi, istilah-istilah yang digunakan dalam penelitian.

2.1 Konsep Dasar Teori Graf

2.1.1 Graf

Graf merupakan representasi dari suatu masalah yang digambarkan sebagai sekumpulan titik (*vertex*) yang dihubungkan dengan sekumpulan garis (*edge*) (Afrianto dan Jamilah, 2012).

Suatu graf G (V,E) terdiri dari dua bagian yaitu sebagai berikut.

(i) Suatu himpunan V=V(G) yang memiliki elemen-elemen yang dinamakan *vertex*, titik atau *node*, dengan V≠Ø.

(ii) Suatu kumpulan E=E(G) yang merupakan pasangan terurut dari titik-titik yang berbeda dinamakan *edge* atau garis (Lipschutz dan Lipson, 2002).

Macam macam graf dilihat dari strukturnya ada 6 macam graf yaitu sebagai berikut:

1. *Multigraph*

   Multigraph adalah graf yang mempunyai satu atau lebih pasangan garis yang menghubungkan 2 titik.

2. *Pseudograph*

   Pseudograph adalah graf yang memiliki satu atau lebih pasangan garis yang menghubungkan 2 titik (*multigraph*) dan memiliki satu atau lebih *loop* pada satu titik tersebut.

3. *Null Graph*

   Null Graph adalah graf yang hanya terdiri dari satu titik.

4. Graf Lengkap

   Graf lengkap adalah graf yang setiap pasang titiknya terdapat garis yang menghubungkannya.

5. Graf Teratur

   Graf Teratur adalah graf yang setiap titiknya mempunyai garis yang menempel padanya dengan jumlah yang sama.

6. Bipartit Graf

   Bipartit Graf adalah graf yang himpunan titiknya dapat dibagi dua yaitu $V = V_1 \cup V_2,\ V_1 \cap V_2 = 0$, dan $\forall_{eij} \in E, i \in V_1, j \in V_2$ (Wibisono,2010).

Hubungan atau lintasan antar titik dalam sebuah grap dapat dibedakan menjadi beberapa jenis, sebagai berikut.

1. *Walk*

   Walk adalah suatu barisan berhingga yang terdiri dari garis dan titik bergantian, dimana tidak ada garis yang digunakan lebih dari satu kali. *Walk* yang diawali dan diakhiri dengan titik yang sama disebut *walk* tertutup,

sedangkan yang diakhiri dan diawali dengan titik yang berbeda disebut *walk* terbuka.

2. Path

Path merupakan *walk* terbuka dimana tidak ada titik yang digunakan lebih dari sekali (Zayadi, 2010).

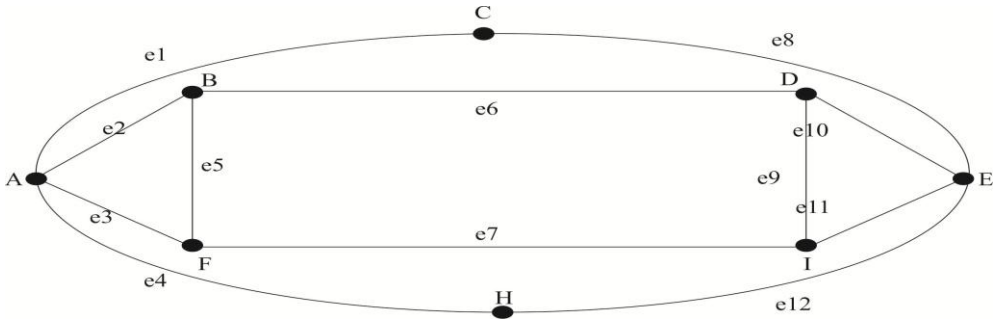
2.1.2 Labeled Graph

Sebuah graf G dikatakan berlabel jika garis dan atau titik-titiknya diberikan berupa data (Lipschutz dan Lipson, 2002). Dalam menggambarkan sebuah logika suatu kejadian suatu graf sering kali diberi label/ bobot, graf demikian disebut *labeled graph* (Wibisono, 2010).

2.1.3 Derajat atau *Degree*

Menurut Wibisono (2010), suatu titik dalam graf dapat mempunyai 1 atau lebih garis yang terhubung padanya atau tidak ada satupun garis yang terhubung padanya. Derajat suatu titik adalah banyaknya garis yang terhubung pada titik tersebut. Titik ganjil adalah titik yang derajatnya ganjil. Titik genap adalah titik yang derajatnya genap.

Contoh berikut adalah graf yang berbeda-beda derajat pada titiknya



Gambar 2.1 Contoh graf yang berbeda-beda derajat pada titiknya

Derajat titiknya adalah:

deg (A) = 4 deg (E) = 4

deg (B) = 3 deg (F) = 3

deg (C) = 2 deg (H) = 2

deg (D) = 3 deg (I) = 3

Jumlah *degree* = 24

Jumlah garis = 12

Jumlah *degree* = 2 kali jumlah garis

2.2 Graf Berarah (*Directed Graph*)

Suatu graf berarah atau *digraph* G terdiri dari.

(i) Suatu himpunan V=V(G) yang elemen-elemennya disebut titik atau garis.

(ii) Suatu E=E(G) dari pasangan-pasangan titik terurut yang disebut busur atau garis berarah atau garis.

Graf berarah G dikatakan berhingga jika himpunan V dari titik-titiknya dan himpunan E dari busurnya (garis berarah) berhingga (Lipschutz dan Lipson, 2002).

2.3 *Indegree* dan *Outdegree*

Derajat sebuah titik pada graf berarah dapat dibagi menjadi dua, sebagai berikut.

a) In degree

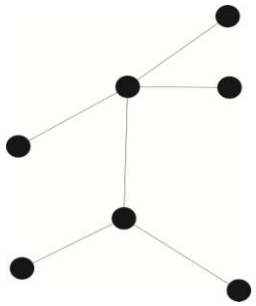
b) Out degree

Indegree suatu titik adalah jumlah garis yang masuk ke suatu titik.

Outdegree suatu titik adalah jumlah garis yang keluar dari suatu titik. Titik yang *indegree*-nya = 0 disebut sumber/asal/*source*. Titik yang *outdegree*-nya = 0 disebut tujuan/*sink* (Wibisono, 2010).

2.4 Pohon (*Tree)*

Pohon adalah suatu graf yang mempunyai n titik, n-1 garis dan tidak mempunyai lingkaran (*cycle free*) serta merupakan graf terhubung. Contoh *tree* dapat dilihat pada Gambar 2.2.



Gambar 2.2 Pohon atau *tree*

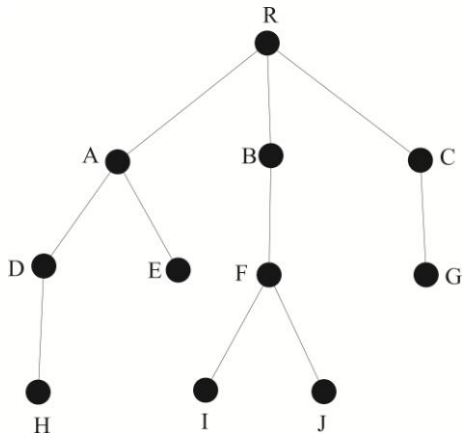
Diagram pohon dapat digunakan sebagai alat untuk memaparkan logika suatu persoalan dengan menggambarkan semua alternatif pemecahannya (Wibisono, 2010).

Menurut Liu (1995), pohon atau *tree* didefinisikan sebagai suatu graf tak berarah yang terhubung (*connected undirected graph*) yang tidak mengandung sirkuit. Sifat-sifat yang dimiliki pohon atau *tree* adalah sebagai berikut.

a) Di dalam suatu pohon, lintasan antara dua titik bersifat tunggal (unik, khas).
b) Di dalam sebuah pohon, jumlah titik satu lebih banyak daripada jumlah garis (e = n-1).
c) Pohon dengan satu atau lebih titik memiliki sedikitnya dua daun.

2.5 Pohon Berakar (*Rooted Tree*)

Suatu *tree* berakar T terdiri dari sebuah *tree* bersama-sama dengan sebuah titik berpola (berbentuk) R yang disebut akar dari *tree*. Pada gambar berikut adalah diagram dari suatu *tree* berakar dimana akar R muncul pada puncak *tree* sesuai Gambar 2.3.



Gambar 2.3 *Rooted tree*
(Lipschutz dan Lipson, 2002).

Menurut Wibisono (2010), sama seperti pohon, dalam graf juga mempunyai akar, cabang, daun. Akar pohon adalah titik yang *indegree*-nya nol (titik sumber). Setiap titik dapat dianggap atau dijadikan akar, titik yang dianggap sebagai akar ditandai dengan lingkaran yang mengelilingi titik tersebut. Daun pohon adalah setiap titik (bukan akar) yang *indegree*-nya 1 dan *outdegree*nya 0. Tinggi pohon adalah panjang garis maksimal dari akar sampai daun.

2.6 Encode, Decode, dan Blob *Code*

2.6.1 Blob *Code*

Menurut Zayadi (2010), Blob *code* merupakan metode pengkodean *tree* dengan melakukan pengumpulan titik-titik dari suatu *tree* yang diberikan. Titik-titik tersebut dikumpulkan ke dalam suatu himpunan yang dilingkarkan yang disebut

Blob. Metode Blob *Code* mempertimbangkan *path* dari suatu titik untuk menentukan kode yang ditampilkan. *Path* (x) didefinisikan sebagai berikut: *path* (x) = (x,*succ*(x), *succ(succ*(x)),…,0).

2.6.2 Encode

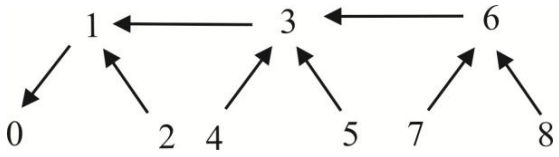
Encode adalah suatu metode untuk melakukan pengkodean ulang terhadap suatu data, sehingga data tersebut apabila dilihat tidak akan sama dengan aslinya, dengan tanpa merusak informasi di dalamnya. Hal ini digunakan agar data yang dikirim tidak bisa diketahui oleh pihak yang tidak berkepentingan (Zayadi, 2010).

2.6.3 Decode

Decode adalah suatu metode pembacaan suatu data yang telah dikirimkan dalam bentuk sandi, kemudian diterjemahkan kembali ke dalam bentuk aslinya (Zayadi, 2010).

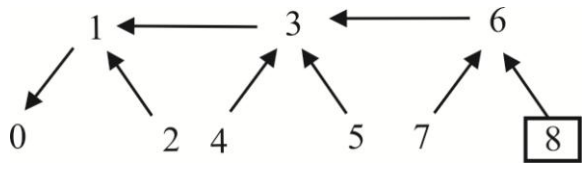
Berikut merupakan contoh *encode tree* (Zayadi, 2010).

Diberikan *tree* dengan 9 titik dan 8 garis.

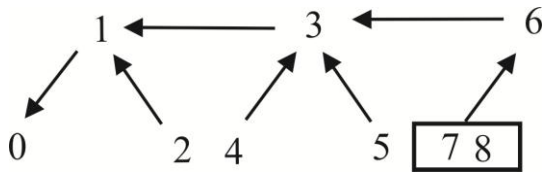


Langkah-langkah untuk memperoleh kode Blob dari *tree* di atas adalah sebagai berikut.

1. Tentukan kode awal B= ( ),

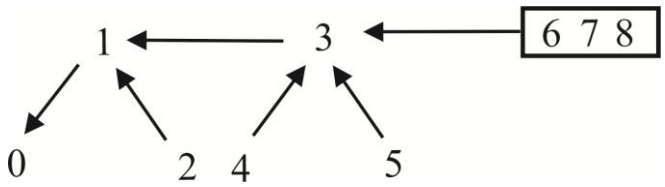


2. Tentukan titik dengan *index* terbesar (n) adalah 8. Kemudian, tentukan *path* (7) = (7,6,3,1,0) dan Blob = (8). Karena *path* (7) ∩ Blob =∅ maka kode yang tampil B= (*succ(*Blob*),* B), yaitu B = (6). Pada konstruksi *tree* yang baru, garis Blob dibuang, lalu tambahkan *edge* Blob→ *succ*(7). Kemudian, buang garis (7) dan terakhir bentuk Blob = Blob ∪ {7},



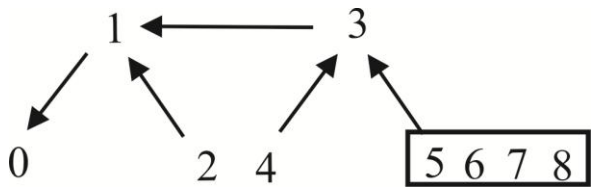
Kode sementara = (6)

3. Tentukan *path* (6) = (6,3,1,0) dan Blob = (7,8). Karena *path* (6) ∩ Blob = ∅, maka kode yang tampil B = (*succ(*Blob*),* B), yaitu B = (6,6). Pada konstruksi *tree* yang baru, garis Blob dibuang, lalu tambahkan garis Blob → *succ*(6). Kemudian, buang garis (6) dan terakhir bentuk Blob = Blob ∪ {6},



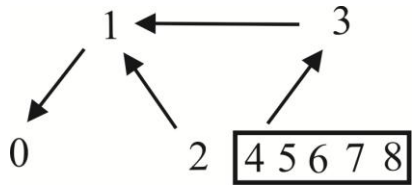
Kode sementara = (6,6)

4. Tentukan *path* (5) = (5,3,1,0) dan Blob = (6,7,8). Karena *path* (5) ∩ Blob = ∅, maka kode yang tampil B = (*succ(*Blob*),* B), yaitu B = (3,6,6). Pada konstruksi *tree* yang baru, garis Blob dibuang, lalu tambahkan garis Blob → *succ*(5). Kemudian, buang garis (5) dan terakhir bentuk Blob = Blob ∪ {5},



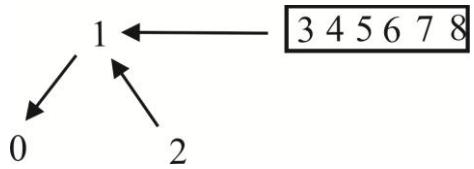
Kode sementara = (3,6,6)

5. Tentukan *path* (4) = (4,3,1,0) dan Blob = (5,6,7,8). Karena *path* (4) ∩ Blob = ∅, maka kode yang tampil B = (*succ(*Blob*)*, B), yaitu B = (3,3,6,6). Pada konstruksi *tree* yang baru, garis Blob dibuang, lalu tambahkan garis Blob → *succ*(4). Kemudian, buang garis (4) dan terakhir bentuk Blob = Blob ∪ {4},



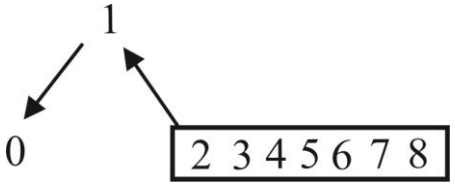
Kode sementara = (3,3,6,6)

6. Tentukan *path* (3) = (3,1,0) dan Blob = (4,5,6,7,8). Karena *path* (3) ∩ Blob = ∅, maka kode yang tampil B = (*succ(*Blob*)*, B), yaitu B = (3,3,3,6,6). Pada konstruksi *tree* yang baru, garis Blob dibuang, lalu tambahkan garis Blob → *succ*(3). Kemudian, buang garis (3) dan terakhir bentuk Blob = Blob ∪ {3},



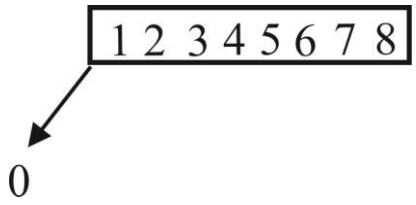
Kode sementara = (3,3,3,6,6)

7. Tentukan *path* (2) = (2,1,0) dan Blob = (3,4,5,6,7,8). Karena *path* (2) ∩ Blob = ∅, maka kode yang tampil B = (*succ(*Blob*)*, B), yaitu B = (3,3,3,6,6). Pada konstruksi *tree* yang baru, garis Blob dibuang, lalu tambahkan garis Blob → *succ*(2). Kemudian, buang garis (2) dan terakhir bentuk Blob = Blob ∪ {2},



Kode sementara = (1,3,3,3,6,6)

8. Tentukan *path* (1) = (1,0) dan Blob = (2,3,4,5,6,7,8). Karena *path* (1) ∩ Blob = ∅, maka kode yang tampil B = (*succ(*Blob*)*, B), yaitu B = (1,3,3,3,6,6). Pada konstruksi *tree* yang baru, garis Blob dibuang, lalu tambahkan garis Blob → *succ*(1). Kemudian, buang garis (1) dan terakhir bentuk Blob = Blob ∪ {1},



B = (1,1,3,3,3,6,6)

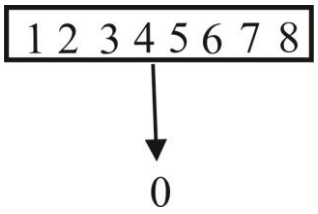
9. Proses *encode* selesai dan diperoleh B = (1,1,3,3,3,6,6).

Berikut merupakan contoh *decode tree* (Zayadi, 2010).

Diberikan kode B = (1,1,3,3,3,6,6)

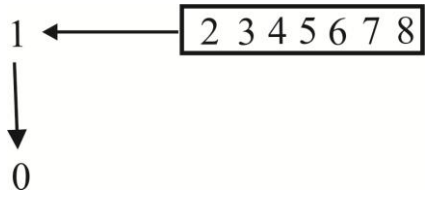
Langkah-langkah untuk memperoleh konstruksi *tree* adalah sebagai berikut.

1. Tentukan konstruksi Blob → 0 dengan n = 7+1,



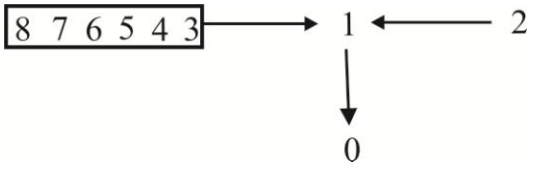
B = (1,1,3,3,3,6,6)

2. Tentukan *path* (1) = (1,0) dan Blob = (2,3,4,5,6,7,8). Karena *path* (1) ∩ Blob = ∅, maka tambahkan garis 1 → *succ(*Blob*)*, buang garis *succ(*Blob*)*, lalu tambahkan garis Blob → b_1, yaitu garis Blob → 1. Kemudian buang angka 1 pada B, sehingga B = (1,3,3,3,6,6),



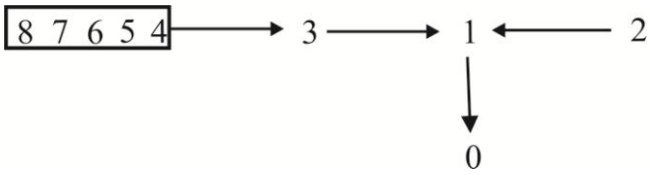
B = (1,3,3,3,6,6)

3. Tentukan *path* (1) = (1,0) dan Blob = (3,4,5,6,7,8). Karena *path* (1) ∩ Blob = ∅, maka tambahkan garis 2 → 1, buang garis Blob → 1, lalu tambahkan garis Blob → b_1, yaitu garis Blob → 1. Kemudian buang angka 1 pada B, sehingga B = (3,3,3,6,6),



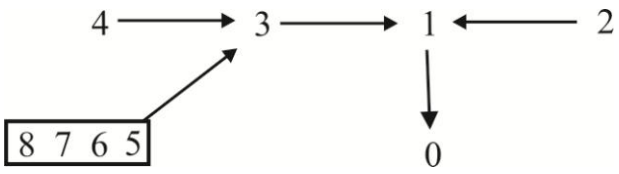
B = (3,3,3,6,6)

4. Tentukan *path* (3) = (3,1,0) dan Blob = (4,5,6,7,8). Karena *path* (3) ∩ Blob = ∅, maka tambahkan garis 3 → 1, buang garis Blob → 3, lalu tambahkan garis Blob → b_1, yaitu garis Blob → 3. Kemudian buang angka 3 pada B, sehingga B = (3,3,6,6),



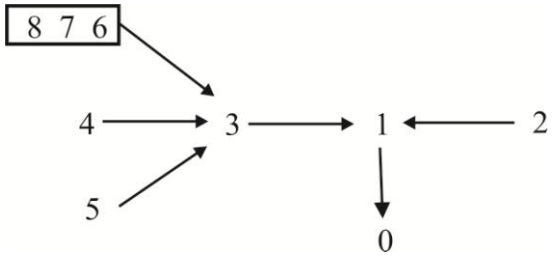
B = (3,3,6,6)

5. Tentukan *path* (3) = (3,1,0) dan Blob = (5,6,7,8). Karena *path* (3) ∩ Blob = ∅, maka tambahkan garis 4 → 3, buang garis Blob → 3, lalu tambahkan garis Blob → b_1, yaitu garis Blob → 3. Kemudian buang angka 3 pada B, sehingga B = (3,6,6),



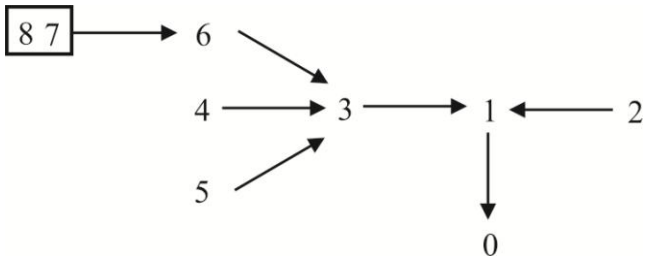
B = (3,6,6)

6. Tentukan *path* (3) = (3,1,0) dan Blob = (6,7,8). Karena *path* (3) ∩ Blob = ∅, maka tambahkan garis 5 → 3, buang garis Blob → 3, lalu tambahkan garis Blob → b_1, yaitu garis Blob → 3. Kemudian buang angka 3 pada B, sehingga B = (6,6),



B = (6,6)

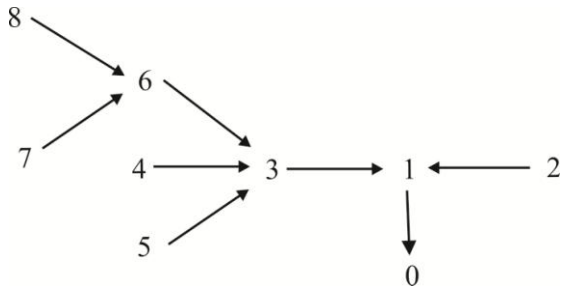
7. Tentukan *path* (6) = (6,3,1,0) dan Blob = (7,8). Karena *path* (6) ∩ Blob = ∅, maka tambahkan garis 6 → 3, buang garis Blob → 3, lalu tambahkan garis Blob → b_1, yaitu garis Blob → 6. Kemudian buang angka 6 pada B, sehingga B = (6),



B = (6)

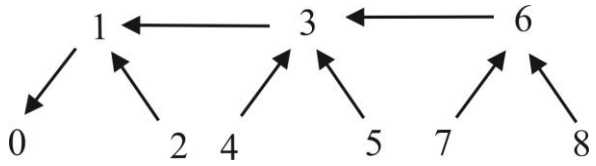
8. Tentukan *path* (6) = (6,3,1,0) dan Blob = (8). Karena *path* (6) ∩ Blob = ∅, maka tambahkan garis 7 → 6, buang garis Blob → 6, lalu tambahkan garis

Blob → b_1, yaitu garis Blob → 6. Kemudian buang angka 6 pada B, sehingga B = ( ),



B = ( )

9. Diperoleh *original tree* berikut.



2.7 Extreme Programming

Menurut Widodo dan Subekti (2006) saat ini metode yang digunakan dalam pengembangan sistem udah cukup banyak berkembang, macam-macam metode pengembangan sistem adalah:

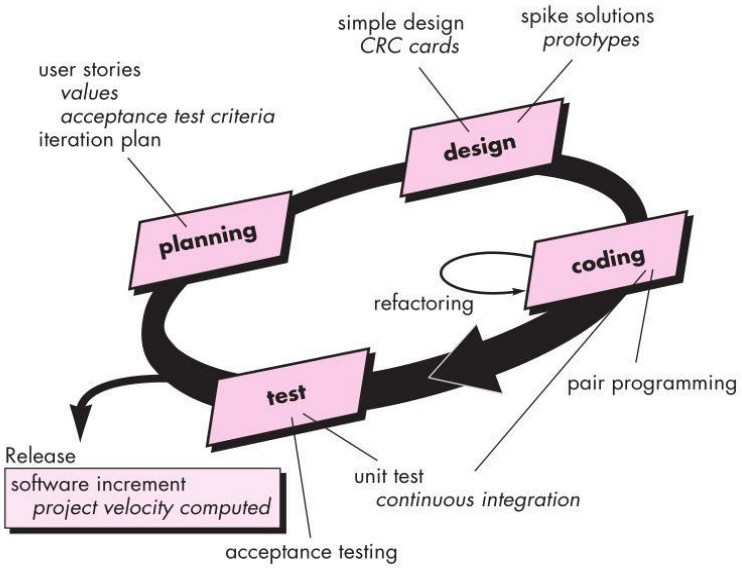
1. *eXtreme Programming (XP)*
2. *Scrum Methodology*
3. *Crystal Family*
4. *Dynamic Systems Development Method (DSDM)*
5. *Adaptive Software Development (ASD)*
6. *Feature Driven Development (FDD)*

Salah satu model yang umum digunakan dalam *agile methods* adalah *extreme programing (XP).* Model ini merupakan metode pengembangan perangkat lunak

yang ringan dan dipelopori oleh Kent Beck, Ron Jeffries, dan Ward Cunningham. XP merupakan *agile methods* yang paling banyak digunakan dan menjadi suatu pendekatan yang sangat terkenal. Sasaran XP adalah tim yang dibentuk berukuran antara kecil sampai sedang saja, tidak perlu menggunakan sebuah tim yang besar. Hal ini dimaksudkan untuk menghadapi *requirements* yang tidak jelas maupun terjadinya perubahan-perubahan *requirements* yang sangat cepat (Widodo dan Subekti, 2006).

Menurut Pressman (2010) terdapat 4 tahapan pada pengembangan perangkat lunak yang menggunakan XP terdiri dari planning seperti memahami kriteria pengguna dan perencanaan pengembangan, *designing* seperti perancangan *prototype* dan tampilan, *coding* termasuk pengintegrasian, dan yang terakhir adalah testing.

Unsur-unsur lain dari *Extreme Programming* meliputi *pair programming* pada tahapan *coding*, pengujian unit semua kode, menghindari fitur-fitur pemrograman sampai mereka benar-benar diperlukan, struktur manajemen yang datar, kesederhanaan dan kejelasan dalam kode, dan seringnya terjadi komunikasi antara programmer dan pelanggan ketika terjadi perubahan kebutuhan pelanggan seiring berlalunya waktu.



Gambar 2.4. Tahapan *Extreme Programming* (Pressman, 2010)