WDMOC Pilot's Guide

The overall purpose of the Whole Disease Model of Oral Cancer (WDMOC) is to simulate the passage of people through the oral cancer care pipeline. Simulated people (referred to as 'entities') start the model with a randomly-generated set of demographic characteristics and disease status. They may or may not undergo regular screening, and may develop oral cancer. Detected cancers (either through screening or through the development of symptoms) are managed according to their stage at diagnosis. Treated cancers may recur, or entities may survive to remission. Entities may die of disease, or of natural causes (i.e., causes unrelated to oral cancer).

The following document outlines the logic underlying the model's structure.

The WDMOC is divided roughly into three types of programs:

1. Global Programs (Glb): These programs perform a function that operates outside a particular entity's pathway through the model.
2. Natural History (NatHist): These programs determine the sequence of disease events an entity will experience in the absence of screening/treatment.
3. System Processes (SysP): These programs represent functions of the health care system.

All programs are subordinate to a top-tier program called the Sequencer (Sequencer.py), which determines the current state of the entity, next event that is scheduled to occur, and what System Process should be run.

The Sequencer (Sequencer.py)

The sequencer is a program that runs all other programs within the WDMOC. Its function can be summarized as follows:

- Specify the number of entities to be sequenced (i.e., the number of individuals you wish to model)
- Create lists to hold a record of all resource utilization and events experienced by the entities ('ResourceList' and 'EventsList' respectively)
- For each entity:
    1. Create the entity (run Glb_CreateEntity.py)
    2. Create a blank list of resources and events for this entity ('resources' and 'events' respectively)
    3. Create a blank list to hold the natural history events ('natHist')
    4. Create some initial characteristics for the entity (Glb_AddInitChars.py)
    5. Sequence the Natural History of the entity (NatHist_OralCancer.py)
    6. Start a loop that:
        - Checks the clock ('allTime') and update the entity's natural history or clinical event (if the entity has a detected cancer) status
        - Identifies the current state ('stateNum') of the entity and performs the appropriate function

       This loop repeats until the entity dies, either of oral cancer or some other cause
    - Record the resources and events the entity has experienced in the 'ResourceList' and 'EventsList' lists.
- Export 'ResourceList' and 'EventsList' to the hard drive – contains records for all entiies, each entity in its own sheet of an array.

0.0 – The Entity (Glb_CreateEntity.py)

The entity represents an individual person. Programmatically, an entity can be thought of as a library that can store a potentially infinite number of variables, which can be created, modified, or deleted at any time. At the time of creation, an entity is imbued with a handful of characteristics that will be used throughout the model run:

- 'currentState' and 'stateNum': variables which describe the current status of the entity (in text and numerically, respectively). The Sequencer uses 'stateNum' to determine what System Process the entity should undergo next.
- 'allTime': a running clock that represents the amount of time (in days) that has elapsed since the entity was created.
- Some variables for the Natural History programs:
    - 'nh_status': serves similar function to 'stateNum' in the natural history process; represents the type of event that is scheduled to occur
    - 'nh_time': serves similar function to 'allTime' in the natural history process; represents the time at which a given event is scheduled to occur
    - 'nh_det': a flag denoting whether or not an oral cancer has been detected (this stops the loop that sequences events)

The model is set up to run multiple entities in succession. The number can be specified in the Sequencer. Entities do not interact with each other.

0.1 – Importing model parameters from Excel sheet (Import_Varnames.py)

The values of all model parameters are stored in an Excel sheet (InputParameter.xlsx). Each parameter has four components that are used by the model:

I. Parameter name (string): a character string identifying the parameter
II. Type number (integer): a number representing the type of number the parameter is, and the type of distribution used to randomly sample its value for a given model iteration

    1 – Percentage/proportion (Beta)
    2 – Continuous value (Gaussian)
    3 – Time to event (Weibull)
    4 – Values that cannot be less than zero (Gamma)
    5 – Percentage/proportions that must sum to 1 (Dirichlet)
    6 – Coefficients from logistic regression (Log odds)

III. Mean value of the parameter
IV. Standard Deviation/Error of the mean

The 'Import_Varnames' program reads and imports these four components from the Excel file using a function from the 'openpyxl' package. It stores the values in a library called 'estimates'. Values can be randomly sampled from the 'estimates' library at any time using the following syntax:

    estimates.*parametername*.sample()

Python will return a randomly sampled value of the parameter.

0.2 – Time-to-event value generation (Glb_GenTime.py)

Some Time-to-event values are handled by a separate process that uses the coefficients of a Weibull regression to generate randomly-sampled values. These values are stored in a separate sheet called 'RegCoeffs' in InputParameter.xslx, with six components that are used by the model:

I. Parameter name (string): a character string identifying the model parameter
II. Factor (string): the variable in the regression (e.g., age, sex, treatment type, etc.)
III. VarType (numeric): A number representing the type of factor:

    1 – Categorical: two or more mutually-exclusive categories; the factor must be in one category
    2 – Continuous: the factor can have any value, which is multiplied by the coefficient to produce the estimate
    3 – Dummy: one or more categories; the factor may have any number of these values (including none)

IV. Value (string): the specific value that each factor may have. Continuous factors may be left blank, as the calculation reads the value from the entity itself, rather than from the Excel file
V. Mean (numeric): the mean value for the coefficient from the regression
VI. SE (numeric): the standard error of the mean of the coefficient

The Glb_GenTime program reads data from the Excel file and the entity to determine which mean and SE values to draw, and then produces a random Weibull estimate of the time to the event based on the individual characteristics of the entity. Values can be randomly sampled by creating an instance of the class 'GenTime':

```
From Glb_GenTime import GenTime
foo = GenTime(estimates, regcoeffs)
foo.sample(entity, 'Parameter Name')
```

Python will return a randomly sampled value of the parameter, based on the values of the entity.

0.3 – The Clock (Glb_Checktime.py)

Events occur over simulated time[1] within the model. The model keeps a running count of the amount of time that has elapsed since an entity's creation using the variable 'allTime'. Time elapses using the simPy package. The model measures time in days.

At the beginning of each loop of the Sequencer, the model runs the 'CheckTime' function. The function does the following:

- Updates the entity's age
- Evaluates whether the entity has died of natural causes
- If the entity is alive and does not have an actively-detected cancer:
    - Loads the entity's natural history (loaded from the 'natHistAr.npy' file) and identifies the time for the next scheduled event. If the time for the event has elapsed, the entity's disease status is updated accordingly, and the program updates the "next event" in the array.
- If the entity *does* have a detected cancer, the clinical history process is updated in a similar way.
    - If the current time is after the scheduled death from disease, the entity dies and the simulation terminates.
    - If the entity is within three months of their scheduled time of death from disease, they are managed in the 'end of life' state ('SysP_TerminalTx.py'), and die once the clock reaches their time of death from oral cancer.
    - If the entity has experienced recurrence, it is updated to the 'incident cancer' state ('SysP_IncidentCancer.py') and may be referred for treatment.

The data I am using gives me time to *detected* recurrence, not the date at which recurrence 'naturally' occurs. As a result, the model assumes that recurrences are detected immediately, even though the reality is that they can theoretically persist undetected for some time.

[1] From here on, any reference to "time" should be read as simulated time. This can be contrasted with 'system time', which is the number of seconds/minutes it takes for the computer to perform a task.

0.4 – Applying Initial Characteristics to an Entity (Glb_ApplyInit.py)

A newly-created entity is assigned some basic demographic characteristics that can be used elsewhere in the model:

- A starting age ('entity.startAge') – random integer between 20 and 60
- Sex – 'Male'or 'Female'
- Smoking status – 'Never' or 'Ever'
- Alcohol use status – 'Heavy' or 'Nonheavy'
- Access to a dentist – binary value (0 – no; 1 – yes)
    - This number is based on the probability that a person has both access to dental care, and that their dentist performs regular screens
- Some useful flags to denote that the entity does not have disease (OPLStatus, hasCancer, diseaseDetected, cancerDetected). These may all be used by other parts of the model.
- A date of death due to natural causes
- A probability of starting the model with OPL – this will be used in the natural history function ('NatHist_DevOPL.py')

0.5 – Cancer Treatment Flags (Glb_CancerFlags)

When an entity's cancer is diagnosed, this program is run to apply some treatment flags to the entity. The program determines whether the entity's cancer is treated with surgery, surgery and radiotherapy, some other treatment (usually chemotherapy + radiation), or left untreated. This eligibility is based on stage at diagnosis.

The entity is assigned a 'prim_txType':

'surgery' – Surgery alone
'surgeryRT' – Surgery with adjuvant radiotherapy
'other' – Any non-surgical management
'notx' – No treatment is given

These categories were derived from treatment records in a retrospective cohort of cancer patients.

This program also creates a variable to count the number of times the entity has been prescribed chemotherapy and radiation ('chemoCount' and 'RTCount' respectively). The probability of receiving RT and/or chemotherapy among the 'other' group is based on probabilities within the same retrospective cohort.

A similar program is run for recurring cancers (Glb_RecurTx), with a different categorization scheme to determine 'recur_txType':

'surgery' – Surgery of any kind
'nonsurgery' – any curative management that does not include surgery
'palliative' – the entity is managed palliatively
'notx' – no additional treatment is given

If the entity has had previous radiation, and they are in the 'nonsurgery' category, they may be re-irradiated. For now, re-irradiation does not affect the entity in any way, but is recorded as it has been shown to be associated with different treatment outcomes.

1.0 – The Natural History of Oral Cancer (NatHist_OralCancer.py)

The program assumes the following trajectory of disease:

- Newly-created entities may or may not have a premalignant disease ('NatHist_DevOPL.py')
- Premalignancies (OPLs) may progress to invasive cancer, or they may spontaneously resolve ('NatHist_OPLProg.py').
- Invasive cancers may progress to a higher stage of disease (I → II → III → IV), or they may be detected symptomatically ('NatHist_UnDet.py').
    - It is possible to die of undetected stage IV cancer

The 'NatHistOCa' program creates a list ('entity.natHist') that holds the timing of each natural history event. The program sequences events based on the 'entity.nh_status' variable. All natural history events are sequenced at this time, using a loop that terminates when the disease is detected through symptoms ('nh_det' = 1), when the OPL resolves on its own, or the entity dies of undetected stage IV cancer. If the entity does not have an OPL, the entity's date of natural death ('entity.natHist_deathAge') is entered into the 'natHist' list as the sole event.

When the program finishes (i.e., the entity's events are sequenced and the loop terminates), the 'natHist' list containing the sequence of events is saved to the hard drive as an array ('natHistAr.npy'), where it can be called by the 'Glb_CheckTime' program.

1.1 – Premalignant Disease Prevalence (NatHist_DevOPL.py)

Newly-created entities may start the simulation with an undiagnosed OPL. Entities with an OPL are assigned a risk category (high, medium, or low) drawn from a Dirichlet distribution.

1.2 – Premalignant Progression to Invasive Cancer (NatHist_OPLProg.py)

This program draws a random sample for the time to progression from OPL to invasive cancer – this time is dependent on the risk score that was assigned in 'NatHist_DevOPL.py'. It also draws a random sample for the time to spontaneous resolution of OPL. Whichever time value is smallest is entered as the next event to occur in the 'entity.natHist' list.

1.3 – Progression of Undetected Oral Cancer (NatHist_UnDet.py)

A cancer at a given stage may do one of two things: 1) progress to a higher stage, or 2) be detected symptomatically. The program determines which of these two things will happen by drawing random time values for each. The lower value is chosen as the next event, and is entered in the 'entity.natHist' list.

Entities with Stage IV cancers that are not detected symptomatically will die of their disease (unless it is detected through some other method like screening).

2.0 – The Dental Screening system process (SysP_ScreenAppt)

Entities with access to a screening dentist will be seen at regular intervals for a dental checkup. If the entity has developed a premalignant lesion, it may be detected in a routine exam. Entities with a detected lesion are asked to return in three weeks. If the lesion persists beyond at three-week follow-up, the entity is referred to a periodontist for additional scrutiny. The periodontist will perform a biopsy of any lesion that is deemed suspicious for premalignancy. Malignant lesions are detected in this way, and referred for pre-malignant management. Non-malignant lesions and lesions that resolve within the three-week period are returned for routine dental checkups.

It is possible for the screening procedure to return a false negative (i.e., the entity *has* premalignant disease but a negative test), in which case they will not be re-screened until the next screening appointment and their disease may progress.

Premalignancies are assigned a grade (low-grade or high-grade), and cancers are assigned treatment characteristics (i.e., 'Glb_CancerFlags.py' is run) at this time.

3.0 – The Premalignancy Management System Process (SysP_OPLManage.py)

Entities with a diagnosed premalignancy undergo regular screening tests. Biopsies are performed at regular intervals (‘entity.OPLscreenInt’). If the entity’s premalignancy has progressed to invasive cancer, it will be detected at a follow-up appointment. Detected cancers are referred for diagnostic workup and treatment.

4.0 – The Invasive Cancer Treatment System Process (SysP_IncidentCancer.py)

Newly-diagnosed entities (or entities with a newly-discovered recurrence) are referred for treatment based on the stage of their disease. Entities whose cancers were detected symptomatically receive a biopsy at this point, whereas entities whose cancers are detected through screening have already received a biopsy.

New cancers are managed in the same way by the stage-appropriate program (SysP_*Stage*Tx – either 'StageOne', 'StageTwo', or 'StageAdv', which is a combination of stage III and IV cancers):

- The entity receives some treatment based on their treatment type ('entity.prim_txType' – determined in the 'Glb_CancerFlags.py' program).
- Based on the treatment received, the entity is assigned a date of recurrence and a date of death due to disease.
- Recurrent disease ('SysP_RecurTx.py) works slightly differently:
    - Entities are flagged for treatment in a similar way to 'Glb_CancerFlags.py'. A separate set of secondary recurrence treatments may be assigned ('entity.recur_txType').
    - If the entity receives either palliative or no treatment, it is assigned a time due to death from disease and referred to the Terminal disease state ('SysP_TerminalTx.py')
    - All other entities receive treatment according to the flag they have been assigned, and the time to second recurrence ('TSR') and time to death from disease are randomly re-sampled

5.0 – Post-Treatment Followup (SysP_Followup.py)

After treatment, entities are seen for regular follow-up appointments. At each appointment, they are screened for recurrence. The intervals between these appointments are determined by the length of time since the end of treatment. Once an entity has been recurrence-free for ten years, it is considered to be in full remission and will not interact with the model again until it dies of natural causes.

N.B. The data that the WDMOC uses to determine the time to recurrence is the time to *detected* recurrence. Since recurrences are detected either symptomatically or at a follow-up appointment, and since it is not possible to distinguish between these two types of detection, the model assumes that an entity's recurrence is detected immediately. As a necessary consequence, recurrences will not be detected through the follow-up process.

I coded this process nevertheless for two reasons. First, entities entering remission will generate followup resource utilization that must be factored into the overall cost of cancer. Second, if this model is adapted to use data from, for example, a clinical trial or another source where "time to detection through regular follow-up appointments" can be distinguished from "time to biological recurrence", this detection process will be relevant.

6.0 – Terminal Disease Management (SysP_TerminalTx.py)

This component of the model simulates the experience of two types of advanced cancers.

1 – Cancers that have received palliative care or who are not being treated:

- The clock advances a month, and the appropriate resource (either palliative or non-treated) is applied to the entity.
- A count of the number of months increases by one, and the clock advances one month
- If the number of months is greater than 520 (i.e., if ten years has elapsed and the entity is still alive) they are in full remission and are referred to the appropriate state

2 – Cancers that are three months from death or less

- The entity receives some form of end-of-life treatment, and the clock advances one month.