🌱 Comprehensive Phenology Model - Crop Physiologist Expert Guide

🔬 Physiological Foundation of Plant Development

Understanding Phenology: The Science of Plant Development

As a crop physiologist, phenology represents the temporal orchestration of plant development - from seed germination through senescence. It's the biological clock that coordinates growth, morphogenesis, and reproduction with environmental cues. Understanding phenology is crucial for predicting harvest timing, optimizing growing conditions, and maximizing yield.

Fundamental Concepts

1. **Thermal Time (Growing Degree Days)**
   - **Principle**: Development rate is temperature-dependent
   - **Base temperature**: Minimum temperature for development
   - **Optimal temperature**: Maximum development rate
   - **Accumulation**: Daily thermal units drive stage transitions
2. **Photoperiod Response**
   - **Day-neutral plants**: Development independent of daylength (lettuce)
   - **Short-day plants**: Flower when days shorten
   - **Long-day plants**: Flower when days lengthen
   - **Critical photoperiod**: Threshold daylength for response
3. **Vernalization**
   - **Cold requirement**: Extended cold period for flowering
   - **Molecular basis**: Epigenetic regulation of flowering genes
   - **Quantitative**: Gradual accumulation of cold units
4. **Developmental Plasticity**
   - **Environmental modulation**: Stress affects development rates
   - **Resource allocation**: Nutrient/water status influences transitions
   - **Hormonal control**: Gibberellins, cytokinins, auxins coordinate development

📊 Mathematical Framework

Thermal Time Calculation

python

```python
def calculate_thermal_time(daily_temperatures, base_temp=4.0, optimal_temp=25.0, max_temp=35.0):
    """
    Calculate thermal time using multiple temperature response functions

    Methods:
    1. Linear: GDD = max(0, T_avg - T_base)
    2. Triangular: Accounts for optimal and maximum temperatures
    3. Beta function: More realistic curvilinear response
    """

    thermal_time_methods = {}

    for day, temp in enumerate(daily_temperatures):
        # Method 1: Simple linear accumulation
        linear_gdd = max(0, temp - base_temp)

        # Method 2: Triangular response (plateau between optimal and max)
        if temp <= base_temp:
            triangular_gdd = 0
        elif temp <= optimal_temp:
            triangular_gdd = (temp - base_temp) * (optimal_temp - base_temp) / (optimal_temp - base_temp)
        elif temp <= max_temp:
            triangular_gdd = (optimal_temp - base_temp) * (max_temp - temp) / (max_temp - optimal_temp)
        else:
            triangular_gdd = 0  # Too hot for development

        # Method 3: Beta function (most realistic)
        beta_gdd = calculate_beta_function_gdd(temp, base_temp, optimal_temp, max_temp)

        thermal_time_methods[day] = {
            'temperature': temp,
            'linear_gdd': linear_gdd,
            'triangular_gdd': triangular_gdd,
            'beta_gdd': beta_gdd
        }

    return thermal_time_methods

def calculate_beta_function_gdd(temperature, t_base, t_opt, t_max):
    """
    Beta function for more realistic temperature response

    Based on research: Wang & Engel (1998), Yin et al. (1995)
```

```
    """
    if temperature <= t_base or temperature >= t_max:
        return 0

    # Beta function parameters
    alpha = 2.0  # Shape parameter
    beta_param = 2.0  # Shape parameter

    # Normalized temperature
    t_norm = (temperature - t_base) / (t_max - t_base)
    t_opt_norm = (t_opt - t_base) / (t_max - t_base)

    # Beta function calculation
    beta_value = (
        (t_norm ** alpha) *
        ((1 - t_norm) ** beta_param) *
        ((1 + alpha + beta_param) ** (alpha + beta_param)) /
        ((alpha ** alpha) * (beta_param ** beta_param))
    )

    # Scale to optimal temperature response
    max_beta = (
        (t_opt_norm ** alpha) *
        ((1 - t_opt_norm) ** beta_param) *
        ((1 + alpha + beta_param) ** (alpha + beta_param)) /
        ((alpha ** alpha) * (beta_param ** beta_param))
    )

    return (t_opt - t_base) * (beta_value / max_beta) if max_beta > 0 else 0
```

Lettuce Growth Stages (CROPGRO-style)

```python
```

```python
from enum import Enum

class LettuceGrowthStage(Enum):
    """Comprehensive lettuce growth stages following CROPGRO paradigm"""

    # Germination and emergence
    GERMINATION = "GE"          # Seed imbibition to radicle emergence
    EMERGENCE = "VE"            # Cotyledon emergence above soil/media

    # Vegetative stages (V1-V15)
    FIRST_LEAF = "V1"           # First true leaf unfolded
    SECOND_LEAF = "V2"          # Second true leaf unfolded
    THIRD_LEAF = "V3"           # Third true leaf unfolded
    FOURTH_LEAF = "V4"          # Fourth true leaf unfolded
    FIFTH_LEAF = "V5"           # Fifth true leaf unfolded
    SIXTH_LEAF = "V6"           # Sixth true leaf unfolded
    SEVENTH_LEAF = "V7"         # Seventh true leaf unfolded
    EIGHTH_LEAF = "V8"          # Eighth true leaf unfolded
    NINTH_LEAF = "V9"           # Ninth true leaf unfolded
    TENTH_LEAF = "V10"          # Tenth true leaf unfolded
    ELEVENTH_LEAF = "V11"       # Eleventh true leaf unfolded
    TWELFTH_LEAF = "V12"        # Twelfth true leaf unfolded
    THIRTEENTH_LEAF = "V13"     # Thirteenth true leaf unfolded
    FOURTEENTH_LEAF = "V14"     # Fourteenth true leaf unfolded
    FIFTEENTH_LEAF = "V15"      # Fifteenth true leaf unfolded

    # Head formation (for heading varieties)
    HEAD_INITIATION = "HI"      # Begin head formation
    HEAD_FORMATION = "HF"       # Active head development
    HEAD_DEVELOPMENT = "HD"     # Head filling and expansion
    HEAD_MATURITY = "HM"        # Head physiologically mature

    # Reproductive stages (for seed production)
    BOLTING = "BO"              # Stem elongation begins
    FLOWER_INITIATION = "FI"    # Flower primordia formation
    FLOWERING = "FL"            # Anthesis
    SEED_FILL = "SF"            # Seed development
    PHYSIOLOGICAL_MATURITY = "PM" # Seeds mature

    # Senescence
    SENESCENCE = "SE"           # Natural senescence begins

@dataclass
```

```python
class StageParameters:
    """Parameters for each growth stage"""

    stage: LettuceGrowthStage
    thermal_requirement: float   # GDD needed to complete stage
    duration_range: Tuple[float, float]  # Min, max duration in days
    photoperiod_sensitivity: float  # 0-1, sensitivity to daylength
    vernalization_requirement: float  # Cold units needed
    stress_sensitivity: float    # 0-1, sensitivity to environmental stress

    # Physiological characteristics
    growth_priority: str        # 'root', 'shoot', 'leaf', 'reproductive'
    relative_growth_rate: float  # Relative to maximum
    sink_strength: float        # Competition for assimilates

    # Morphological features
    typical_leaf_number: int     # Expected leaf number at stage end
    node_appearance_rate: float  # Nodes per GDD
    specific_leaf_area: float    # m²/g leaf dry weight

def initialize_stage_parameters():
    """Initialize parameters for all lettuce growth stages"""

    return {
        LettuceGrowthStage.GERMINATION: StageParameters(
            stage=LettuceGrowthStage.GERMINATION,
            thermal_requirement=45.0,
            duration_range=(2, 5),
            photoperiod_sensitivity=0.0,
            vernalization_requirement=0.0,
            stress_sensitivity=0.9,  # Very sensitive during germination
            growth_priority='root',
            relative_growth_rate=0.3,
            sink_strength=0.8,       # High priority for emergence
            typical_leaf_number=0,
            node_appearance_rate=0.0,
            specific_leaf_area=0.0
        ),

        LettuceGrowthStage.EMERGENCE: StageParameters(
            stage=LettuceGrowthStage.EMERGENCE,
            thermal_requirement=35.0,
            duration_range=(1, 3),
            photoperiod_sensitivity=0.0,
```

```
        vernalization_requirement=0.0,
        stress_sensitivity=0.8,
        growth_priority='shoot',
        relative_growth_rate=0.5,
        sink_strength=0.9,
        typical_leaf_number=0,
        node_appearance_rate=0.0,
        specific_leaf_area=25.0
    ),

    LettuceGrowthStage.FIRST_LEAF: StageParameters(
        stage=LettuceGrowthStage.FIRST_LEAF,
        thermal_requirement=40.0,
        duration_range=(2, 4),
        photoperiod_sensitivity=0.1,
        vernalization_requirement=0.0,
        stress_sensitivity=0.7,
        growth_priority='leaf',
        relative_growth_rate=0.7,
        sink_strength=0.8,
        typical_leaf_number=1,
        node_appearance_rate=0.025,  # 1 node per 40 GDD
        specific_leaf_area=28.0
    ),

    # Continue for all stages...
    LettuceGrowthStage.HEAD_FORMATION: StageParameters(
        stage=LettuceGrowthStage.HEAD_FORMATION,
        thermal_requirement=120.0,
        duration_range=(8, 15),
        photoperiod_sensitivity=0.2,
        vernalization_requirement=0.0,
        stress_sensitivity=0.4,
        growth_priority='leaf',
        relative_growth_rate=1.0,  # Peak growth rate
        sink_strength=1.0,         # Maximum sink strength
        typical_leaf_number=12,
        node_appearance_rate=0.0,  # Node formation complete
        specific_leaf_area=22.0
    ),

    LettuceGrowthStage.BOLTING: StageParameters(
        stage=LettuceGrowthStage.BOLTING,
        thermal_requirement=80.0,
```

```
        duration_range=(4, 8),
        photoperiod_sensitivity=0.8,  # Day length sensitive
        vernalization_requirement=0.0,
        stress_sensitivity=0.3,
        growth_priority='reproductive',
        relative_growth_rate=0.6,
        sink_strength=0.9,
        typical_leaf_number=15,
        node_appearance_rate=0.0,
        specific_leaf_area=20.0
    )
}
```

🌡️ Environmental Modifiers

Temperature Effects on Development

```
python
```

```python
def calculate_temperature_development_factor(temperature, stage_params):
    """
    Calculate temperature effect on development rate

    Different stages have different temperature sensitivities:
    - Germination: Sensitive to temperature extremes
    - Vegetative: Broad temperature tolerance
    - Reproductive: Sensitive to high temperatures
    """

    # Base temperature response
    base_factor = calculate_beta_function_gdd(temperature, 4.0, 22.0, 35.0) / 18.0

    # Stage-specific modifications
    stage_name = stage_params.stage.value

    if stage_name in ['GE', 'VE']:  # Germination/emergence
        # More sensitive to temperature extremes
        if temperature < 10 or temperature > 30:
            base_factor *= 0.5
    elif stage_name.startswith('V'):  # Vegetative stages
        # Broad temperature tolerance
        if 15 <= temperature <= 28:
            base_factor *= 1.0
        else:
            base_factor *= 0.8
    elif stage_name in ['BO', 'FI', 'FL']:  # Reproductive stages
        # Sensitive to high temperatures (bolting acceleration)
        if temperature > 25:
            base_factor *= 1.0 + 0.1 * (temperature - 25)  # Accelerated bolting
        elif temperature < 15:
            base_factor *= 0.6

    return max(0.1, min(2.0, base_factor))  # Bound between 0.1 and 2.0

def calculate_photoperiod_effect(daylength, stage_params, latitude=40.0):
    """
    Calculate photoperiod effect on development

    Lettuce is generally day-neutral, but some varieties show responses:
    - Long days can accelerate bolting
    - Short days may delay some developmental processes
    """
```

```python
    photoperiod_sensitivity = stage_params.photoperiod_sensitivity

    if photoperiod_sensitivity == 0.0:
        return 1.0  # No photoperiod effect

    # Critical photoperiod for lettuce (approximate)
    critical_photoperiod = 14.0  # hours

    if stage_params.stage.value in ['BO', 'FI', 'FL']:  # Reproductive stages
        # Long days accelerate bolting
        if daylength > critical_photoperiod:
            photoperiod_factor = 1.0 + photoperiod_sensitivity * (daylength - critical_photoperiod) / 4.0
        else:
            photoperiod_factor = 1.0 - photoperiod_sensitivity * (critical_photoperiod - daylength) / 4.0
    else:
        # Vegetative stages - minimal photoperiod effect
        photoperiod_factor = 1.0 + photoperiod_sensitivity * 0.1 * math.sin(2 * math.pi * daylength / 24)

    return max(0.5, min(1.5, photoperiod_factor))  # Bound between 0.5 and 1.5
```

Stress Effects on Development

python

```python
def calculate_stress_development_effects(stress_factors, stage_params):
    """
    Calculate how environmental stresses affect development rate

    Stress effects vary by developmental stage:
    - Early stages: Stress slows development
    - Later stages: Some stresses can accelerate development (bolting)
    """

    stress_sensitivity = stage_params.stress_sensitivity
    stage_name = stage_params.stage.value

    # Weighted stress calculation
    stress_weights = {
        'water_stress': 0.4,
        'temperature_stress': 0.3,
        'nutrient_stress': 0.2,
        'light_stress': 0.1
    }

    overall_stress = sum(
        stress_factors.get(stress_type, 0) * weight
        for stress_type, weight in stress_weights.items()
    )

    # Stage-specific stress responses
    if stage_name in ['GE', 'VE']:  # Early stages
        # Stress always slows development
        stress_factor = 1.0 - (stress_sensitivity * overall_stress)

    elif stage_name.startswith('V') and int(stage_name[1:]) <= 8:  # Early vegetative
        # Moderate stress slowing
        stress_factor = 1.0 - (stress_sensitivity * overall_stress * 0.7)

    elif stage_name.startswith('V') and int(stage_name[1:]) > 8:  # Late vegetative
        # Some stress may accelerate development (escape response)
        if overall_stress > 0.5:
            stress_factor = 1.0 + (stress_sensitivity * (overall_stress - 0.5))
        else:
            stress_factor = 1.0 - (stress_sensitivity * overall_stress * 0.5)

    elif stage_name in ['HI', 'HF', 'HD']:  # Head formation
        # Head formation sensitive to stress
```

```
        stress_factor = 1.0 - (stress_sensitivity * overall_stress * 1.2)

    elif stage_name in ['BO', 'FI', 'FL']:  # Reproductive stages
        # Stress often accelerates reproductive development
        stress_factor = 1.0 + (stress_sensitivity * overall_stress * 0.8)

    else:
        # Default response
        stress_factor = 1.0 - (stress_sensitivity * overall_stress)

    return max(0.2, min(2.0, stress_factor))  # Bound response
```

🔄 Development Rate Integration

Daily Development Rate Calculation

```
python
```

```python
def calculate_daily_development_rate(temperature, daylength, stress_factors,
                                     current_stage_params, vernalization_status=0.0):
    """
    Integrate all environmental factors to calculate daily development rate

    Development Rate = Base_rate * Temp_factor * Photo_factor * Stress_factor * Vern_factor
    """

    # Base development rate (fraction of stage completed per day)
    thermal_time_today = calculate_beta_function_gdd(temperature, 4.0, 22.0, 35.0)
    base_rate = thermal_time_today / current_stage_params.thermal_requirement

    # Environmental modifiers
    temp_factor = calculate_temperature_development_factor(temperature, current_stage_params)
    photo_factor = calculate_photoperiod_effect(daylength, current_stage_params)
    stress_factor = calculate_stress_development_effects(stress_factors, current_stage_params)

    # Vernalization factor (for cold-requiring stages)
    if current_stage_params.vernalization_requirement > 0:
        vern_factor = min(1.0, vernalization_status / current_stage_params.vernalization_requirement)
    else:
        vern_factor = 1.0

    # Integrated development rate
    development_rate = base_rate * temp_factor * photo_factor * stress_factor * vern_factor

    return {
        'development_rate': development_rate,
        'thermal_time': thermal_time_today,
        'base_rate': base_rate,
        'temperature_factor': temp_factor,
        'photoperiod_factor': photo_factor,
        'stress_factor': stress_factor,
        'vernalization_factor': vern_factor,
        'stage_completion_fraction': development_rate
    }

def update_stage_progress(current_progress, daily_dev_rate, stage_params):
    """
    Update progress through current developmental stage

    Returns True if stage transition should occur
    """
```

```python
    new_progress = current_progress + daily_dev_rate['development_rate']

    # Check for stage completion
    stage_completed = new_progress >= 1.0

    # Calculate days in current stage
    if daily_dev_rate['development_rate'] > 0:
        estimated_days_remaining = (1.0 - new_progress) / daily_dev_rate['development_rate']
    else:
        estimated_days_remaining = float('inf')

    return {
        'new_progress': min(1.0, new_progress),
        'stage_completed': stage_completed,
        'estimated_days_remaining': estimated_days_remaining,
        'development_rate': daily_dev_rate['development_rate']
    }
```

Stage Transition Logic

```python
```

```python
def determine_next_stage(current_stage, variety_type='leaf', environmental_conditions=None):
    """
    Determine the next developmental stage based on current stage and variety type

    Lettuce varieties:
    - 'leaf': Leafy varieties (no head formation)
    - 'butterhead': Loose head formation
    - 'crisphead': Tight head formation
    - 'romaine': Upright growth habit
    """

    stage_transitions = {
        'leaf': {
            LettuceGrowthStage.GERMINATION: LettuceGrowthStage.EMERGENCE,
            LettuceGrowthStage.EMERGENCE: LettuceGrowthStage.FIRST_LEAF,
            LettuceGrowthStage.FIRST_LEAF: LettuceGrowthStage.SECOND_LEAF,
            # ... continue through vegetative stages
            LettuceGrowthStage.FIFTEENTH_LEAF: LettuceGrowthStage.BOLTING,  # Skip head formation
            LettuceGrowthStage.BOLTING: LettuceGrowthStage.FLOWER_INITIATION,
        },

        'butterhead': {
            LettuceGrowthStage.GERMINATION: LettuceGrowthStage.EMERGENCE,
            LettuceGrowthStage.EMERGENCE: LettuceGrowthStage.FIRST_LEAF,
            # ... vegetative stages
            LettuceGrowthStage.TENTH_LEAF: LettuceGrowthStage.HEAD_INITIATION,
            LettuceGrowthStage.HEAD_INITIATION: LettuceGrowthStage.HEAD_FORMATION,
            LettuceGrowthStage.HEAD_FORMATION: LettuceGrowthStage.HEAD_DEVELOPMENT,
            LettuceGrowthStage.HEAD_DEVELOPMENT: LettuceGrowthStage.HEAD_MATURITY,
            LettuceGrowthStage.HEAD_MATURITY: LettuceGrowthStage.BOLTING,
        }
    }

    # Check for environmental triggers (premature bolting)
    if environmental_conditions:
        if (current_stage.value.startswith('V') or
            current_stage in [LettuceGrowthStage.HEAD_INITIATION, LettuceGrowthStage.HEAD_FORMATION]):

            # Check bolting triggers
            if should_trigger_bolting(environmental_conditions):
                return LettuceGrowthStage.BOLTING

    # Normal stage progression
```

```
    transitions = stage_transitions.get(variety_type, stage_transitions['leaf'])
    return transitions.get(current_stage, LettuceGrowthStage.SENESCENCE)

def should_trigger_bolting(environmental_conditions):
    """
    Check if environmental conditions trigger premature bolting

    Bolting triggers:
    - High temperature (>25°C sustained)
    - Long photoperiods (>14 hours)
    - Water stress
    - Root restriction
    - High temperature + long days (synergistic)
    """

    temp_trigger = environmental_conditions.get('average_temp_7day', 20) > 25
    photoperiod_trigger = environmental_conditions.get('daylength', 12) > 14
    stress_trigger = environmental_conditions.get('water_stress', 0) > 0.4

    # Temperature is the strongest trigger
    if environmental_conditions.get('average_temp_7day', 20) > 28:
        return True

    # Combined triggers
    if temp_trigger and photoperiod_trigger:
        return True

    if temp_trigger and stress_trigger:
        return True

    return False
```

🎯 Practical Applications

Harvest Timing Prediction

```
python
```

```python
def predict_harvest_timing(current_stage, stage_progress, environmental_forecast,
                          variety_type='butterhead'):
    """
    Predict optimal harvest timing based on current development and forecast

    Harvest criteria vary by lettuce type:
    - Leaf lettuce: Sufficient leaf area (any time after V8)
    - Head lettuce: Head maturity (HM stage)
    - Processing: Specific timing for quality
    """

    harvest_stages = {
        'leaf': [LettuceGrowthStage.EIGHTH_LEAF, LettuceGrowthStage.FIFTEENTH_LEAF],
        'butterhead': [LettuceGrowthStage.HEAD_MATURITY],
        'crisphead': [LettuceGrowthStage.HEAD_MATURITY],
        'romaine': [LettuceGrowthStage.HEAD_DEVELOPMENT, LettuceGrowthStage.HEAD_MATURITY]
    }

    target_stages = harvest_stages.get(variety_type, harvest_stages['leaf'])

    # Calculate days to each harvest stage
    stage_params = initialize_stage_parameters()
    days_to_harvest = []

    simulated_stage = current_stage
    simulated_progress = stage_progress
    days_elapsed = 0

    while simulated_stage not in target_stages and days_elapsed < 90:  # 90-day limit
        # Simulate daily development using forecast
        daily_temp = environmental_forecast.get(days_elapsed, {}).get('temperature', 20)
        daily_photoperiod = environmental_forecast.get(days_elapsed, {}).get('photoperiod', 12)
        daily_stress = environmental_forecast.get(days_elapsed, {}).get('stress_factors', {})

        current_stage_params = stage_params[simulated_stage]
        daily_dev = calculate_daily_development_rate(
            daily_temp, daily_photoperiod, daily_stress, current_stage_params
        )

        progress_update = update_stage_progress(
            simulated_progress, daily_dev, current_stage_params
        )
```

```
        simulated_progress = progress_update['new_progress']
        days_elapsed += 1

        # Check for stage transition
        if progress_update['stage_completed']:
            simulated_stage = determine_next_stage(simulated_stage, variety_type)
            simulated_progress = 0.0

            # Check if we've reached a harvest stage
            if simulated_stage in target_stages:
                days_to_harvest.append({
                    'stage': simulated_stage,
                    'days': days_elapsed,
                    'harvest_quality': assess_harvest_quality(simulated_stage, variety_type)
                })

    return {
        'harvest_predictions': days_to_harvest,
        'optimal_harvest': min(days_to_harvest, key=lambda x: x['days']) if days_to_harvest else None,
        'latest_harvest': max(days_to_harvest, key=lambda x: x['days']) if days_to_harvest else None
    }

def assess_harvest_quality(harvest_stage, variety_type):
    """Assess expected harvest quality at different stages"""

    quality_matrix = {
        ('leaf', LettuceGrowthStage.EIGHTH_LEAF): {'size': 'small', 'quality': 'tender', 'yield': 'low'},
        ('leaf', LettuceGrowthStage.TWELFTH_LEAF): {'size': 'medium', 'quality': 'good', 'yield': 'medium'},
        ('leaf', LettuceGrowthStage.FIFTEENTH_LEAF): {'size': 'large', 'quality': 'mature', 'yield': 'high'},

        ('butterhead', LettuceGrowthStage.HEAD_DEVELOPMENT): {'size': 'developing', 'quality': 'good', 'yield': 'med
        ('butterhead', LettuceGrowthStage.HEAD_MATURITY): {'size': 'full', 'quality': 'excellent', 'yield': 'high'},
    }

    return quality_matrix.get((variety_type, harvest_stage),
                    {'size': 'unknown', 'quality': 'unknown', 'yield': 'unknown'})
```

Production Planning

python

```python
def calculate_crop_cycle_timing(sowing_date, variety_type, growing_conditions):
    """
    Calculate complete crop cycle timing for production planning

    Outputs:
    - Key milestone dates
    - Resource requirement timing
    - Harvest window planning
    """

    # Initialize simulation
    current_date = sowing_date
    current_stage = LettuceGrowthStage.GERMINATION
    stage_progress = 0.0

    # Track key milestones
    milestones = {
        'sowing': sowing_date,
        'emergence': None,
        'transplant_ready': None,  # V2-V3 stage
        'head_initiation': None,
        'harvest_ready': None,
        'bolting_risk': None
    }

    # Simulate crop development
    stage_params = initialize_stage_parameters()

    for day in range(120):  # 120-day simulation
        # Get daily conditions
        daily_conditions = get_daily_conditions(current_date, growing_conditions)

        # Calculate development
        daily_dev = calculate_daily_development_rate(
            daily_conditions['temperature'],
            daily_conditions['photoperiod'],
            daily_conditions['stress_factors'],
            stage_params[current_stage]
        )

        # Update progress
        progress_update = update_stage_progress(stage_progress, daily_dev, stage_params[current_stage])
        stage_progress = progress_update['new_progress']
```

```
        # Check milestones
        if current_stage == LettuceGrowthStage.EMERGENCE and milestones['emergence'] is None:
            milestones['emergence'] = current_date

        if current_stage in [LettuceGrowthStage.SECOND_LEAF, LettuceGrowthStage.THIRD_LEAF] and milestones[
            milestones['transplant_ready'] = current_date

        if current_stage == LettuceGrowthStage.HEAD_INITIATION and milestones['head_initiation'] is None:
            milestones['head_initiation'] = current_date

        if current_stage == LettuceGrowthStage.HEAD_MATURITY and milestones['harvest_ready'] is None:
            milestones['harvest_ready'] = current_date

        if current_stage == LettuceGrowthStage.BOLTING and milestones['bolting_risk'] is None:
            milestones['bolting_risk'] = current_date

        # Stage transition
        if progress_update['stage_completed']:
            current_stage = determine_next_stage(current_stage, variety_type, daily_conditions)
            stage_progress = 0.0

        current_date += timedelta(days=1)

        # Stop if harvest ready
        if milestones['harvest_ready'] is not None:
            break

    # Calculate cycle statistics
    if milestones['emergence'] and milestones['harvest_ready']:
        total_cycle_days = (milestones['harvest_ready'] - milestones['sowing']).days
        growing_period = (milestones['harvest_ready'] - milestones['emergence']).days
    else:
        total_cycle_days = None
        growing_period = None

    return {
        'milestones': milestones,
        'total_cycle_days': total_cycle_days,
        'growing_period_days': growing_period,
        'current_stage': current_stage,
        'production_efficiency': assess_production_efficiency(milestones, variety_type)
    }
```

```python
def get_daily_conditions(date, base_conditions):
    """Generate daily environmental conditions from base parameters"""

    # Seasonal temperature variation
    day_of_year = date.timetuple().tm_yday
    seasonal_temp_modifier = 5 * math.sin(2 * math.pi * (day_of_year - 80) / 365)

    # Photoperiod calculation
    latitude = base_conditions.get('latitude', 40)
    photoperiod = calculate_photoperiod(day_of_year, latitude)

    return {
        'temperature': base_conditions['base_temperature'] + seasonal_temp_modifier,
        'photoperiod': photoperiod,
        'stress_factors': base_conditions.get('stress_factors', {}),
        'humidity': base_conditions.get('humidity', 70),
        'light_intensity': base_conditions.get('light_intensity', 400)
    }

def calculate_photoperiod(day_of_year, latitude):
    """Calculate photoperiod (daylength) for given day and latitude"""

    # Solar declination
    declination = 23.45 * math.sin(math.radians(360 * (284 + day_of_year) / 365))

    # Hour angle
    hour_angle = math.degrees(math.acos(-math.tan(math.radians(latitude)) * math.tan(math.radians(declination))

    # Photoperiod in hours
    photoperiod = 2 * hour_angle / 15

    return max(8, min(16, photoperiod))  # Bound between 8 and 16 hours
```

This comprehensive phenology model enables precise prediction of crop development, optimizing production timing and resource management in hydroponic systems for maximum efficiency and quality.